The Thesis committee for William Rey Patterson Jr.

Certifies that this is the approved version of the following thesis:

PipeSynth: Automated Topological and Parametric Design of Fluid Networks

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: _____

Matthew I. Campbell

_____

Carolyn C. Seepersad

PipeSynth: Automated Topological and Parametric Design of Fluid Networks

by

William Rey Patterson Jr., B.S.

Thesis

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2010

Abstract

PipeSynth: Automated Topological and

Parametric Design of Fluid Networks

William Rey Patterson Jr., MSE

The University of Texas at Austin, 2010

Supervisor: Matthew I. Campbell

PipeSynth is a design automation approach that combines various optimization research and artificial intelligence methods for synthesizing fluid networks. Starting with only the port locations, PipeSynth generates and optimizes the most effective network for a given application. This ideal network is found by not only optimizing the sizes of each pipe and orientation of fittings in the network (parameters), but also optimizing the layouts of how they are all connected (topology). Using Uniform-Cost-Search for topology optimization, and a combination of non-gradient based parametric optimization methods, PipeSynth demonstrates how advances in automated design can enable engineers to manage much more complex fluid network problems. PipeSynth uses a unique representation of fluid networks that synthesizes and optimizes networks one pipe at a time, in three-dimensional space. PipeSynth has successfully solved several problems containing multiple interlaced networks concurrently with multiple inputs and outputs. PipeSynth shows the power of automated design and optimization in producing solutions more effectively and efficiently than traditional design approaches.

TABLE OF CONTENTS

LIST OF TABLES

TABLE OF FIGURES

1 INTRODUCTION

A fluid network is a set of interconnecting channels that directs the movement of pressure driven fluids. This study focuses on a design approach that automatically synthesizes and optimizes fluid networks known as PipeSynth. Two simple examples of pipe networks include hot and cold water pipes running through the walls and ceilings of a building, and an irrigation system designed to water a field. In the first example, the pipes can be routed in several different ways and not all hot pipes need to reach the same areas as the cold pipes do. In addition, each pipe needs to be able to travel through small openings and crawlspaces using standard pipe fittings. These pipes must also be large enough to adequately transport the fluid without causing too much pressure drop. PipeSynth was developed to handle these challenges, and more, while designing networks that are both inexpensive, and efficient.

Cost and performance are typically the driving factors for optimizing most fluid networks. Therefore, PipeSynth seeks to minimize the cost and the pressure drop across the inlets and outlets of each possible solution it generates. Computational geometry techniques are used to ensure that the pipes have adequate clearance between adjacent pipes and other objects defined in each particular problem. After several possible solutions are found that satisfy all the application's constraints, the best one is presented to the user.

PipeSynth uses a unique representation of fluid networks that allows it to synthesize and optimize networks, one piece at a time. Designing pipe networks with realistic pipe fittings requires the program to consider the geometric constraints of the pipe fittings. For instance, if a pipe coming out of a wall has a 90 degree elbow attached to it, then the next pipe leaving that fitting must actually be perpendicular to the first pipe. PipeSynth determines the location of every pipe in each solution candidate by reconstructing the network starting from input ports, and attaching one pipe at time until the desired network is achieved. This assures that each port, on every fitting, will line up with each straight interconnecting pipe that joins them.

PipeSynth produces realistic solutions by meeting additional design considerations with constraint functions. The first constraint verifies that all the specified inputs and outputs are connected, with no additional open ports are present in the network. The remaining two constraints prevent the pipes from intersecting with each other, and from passing through physical objects, and boundaries.

For network synthesis, PipeSynth focuses on constructing designs using rigid circular pipes and fittings. For pipe flow, the fluid is driven by an imbalance of pressures (Hagen-Poiseuille flow) (1). To maintain a desired flowrate, a large enough pressure must be supplied at the inlet side to overcome the effects of viscous friction causing the fluid to lose momentum(1). Existing fluid network software analysis tools can approximate the pressures and flowrates in fluid networks quite admirably. Effectively, they generally require the user to first provide a layout that describes the network (2)(3). While some of these software tools can perform parametric optimization on parameters such as the pipe diameters, none of these programs can design and optimize a network from scratch, using real pipe fittings.

Another unique feature of PipeSynth is its ability to concurrently construct and optimize multiple networks that share the same physical design space, but have separate, non-interacting fluids. These disjoined networks, such as separate hot and cold water lines, will be referred to as separate channels during the remainder of this report. By synthesizing and optimizing these channels in parallel, this approach assures that the best possible design can be found. Optimizing each channel successively would restrict the possible solutions and make the resulting solutions dependent upon the order in which the channels were optimized.

Every problem has an ideal design best suited for its respective application. Theoretically, this design can be created and can be found. However, it is the design of the search process that determines how this ideal solution can be found. This paper describes, in detail, the individual components of the search process that comprises PipeSynth.

2 BACKGROUND

Nearly every manufacturing facility and building relies on transportation of fluids through pipes. Extensive guides and handbooks have been developed for analyzing and designing pipe networks (4)(5)(6)(7)(8)(9)(10)(11). However, fully automated synthesis and optimization techniques are virtually absent from design.

This first part of this section provides an overview of previous research on analysis and optimization of fluid networks. This is followed by a general background of computation synthesis and topological and parametric optimization methods used to form the design approaches of PipeSynth.

2.1 RELATED WORK

There have been numerous studies on applying optimization techniques to various parts of the network design process. Numerous topology optimization studies have had success when constraining the network joint locations to either a predefined, or randomly generated grid(12)(13)(14)(15)(16)(17). There have been a wide variety of approaches used to optimize fluid networks, everything from direct numerical solutions (16) (18), to genetic algorithms (13)(15)(17) and even simulated annealing (15). However, all these approaches limit the optimization search space by constricting each network junction point to lie on one of 10 to 100 predefined nodes. Then by their specific optimization technique, they determine if and how each of these nodes should be connected to each other. Many of the nodes will be left unused, as long as the inlets (sources) and outputs (sinks) are connected. Other studies have focused on the parametric optimization of existing topologies (19)(18). These papers focused on minimizing cost by balancing the pressure drops between different pipes. These studies controlled the pressure drops within their networks by optimizing the individual pipe diameters. One common aspect in these studies is that they do not account for the cost and geometric constraints encountered when using real pipe fittings.

Many of the same ideas, previously studied for pipe flow analysis, led to the development of commercially available fluid network software tools (2)(3). Compared to traditional computational fluid dynamics (CFD), these software tools do not focus on individual particle flow behavior, rather they primarily consider the behavior of the mean flow throughout

the networks. These commercial pipe flow analysis tools are aimed at large scale civil and agricultural planning where predefined networks (topologies) exist (2)(3). These software packages can also be used to model and evaluate predefined networks to determine if the network layouts can sufficiently handle the flow requirements desired. These programs can easily use brute force numerical methods to solve the fluid dynamics of the network because having a rigid network design does not require multiple topology evaluations. These programs can also assist in other features such as the sizing and placement of pump, tanks, and valves. There are also programs that can parametrically optimize the size of pipes within the network or modify a given topology by adding extra pipes between two existing lines. However, these programs still leave it to the user or installer to decide on the system of fittings that will join the intersection points of the network found by the software analysis tool.

2.2 COMPUTATIONAL SYNTHESIS

Successfully solving any problem efficiently requires a solid design approach. Computational synthesis can be viewed as a cycle, known as the search process that continues until either a feasible solution is found, or the rate of finding better solutions converges to an optimal level as specified by the user. The search process flows through three distinct steps. Figure 1 shows this process, starting with a problem defined by the user and returning a final solution upon completion.

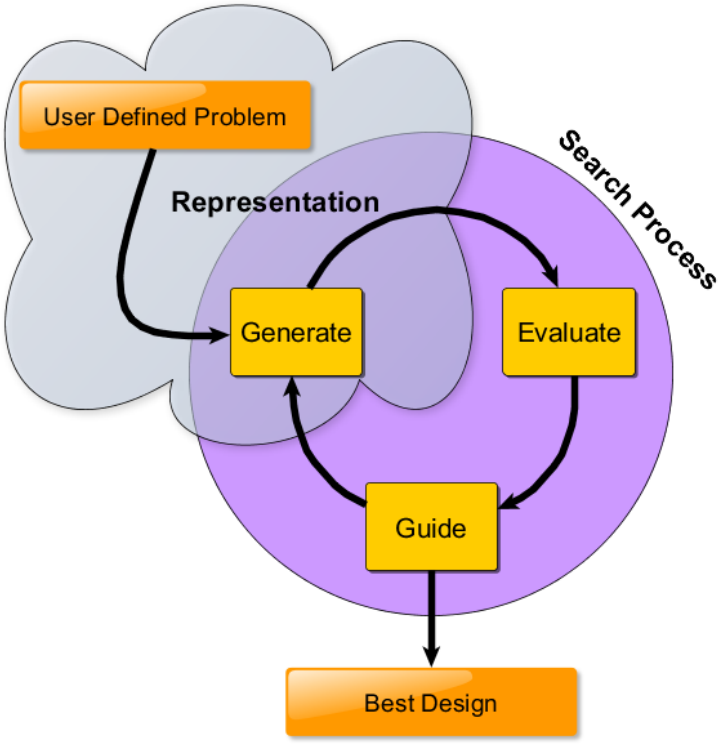

Figure 1. General flowchart for computation synthesis

Integration of the user's problem with the search process requires a common representation as seen in the top left of Figure 1. This representation should take into account how the problem will be solved and optimized. At the very least, the problem must be represented in a form that will allow the computer to be able to generate new potential design solutions (candidates). In addition, there should be enough information represented such that the quality of each potential solution can be evaluated.

With a well-defined representation, the search process begins by generating a new candidate. The generation step begins the search process by modifying the original design given by the user to create a new solution candidate. After each trial, the quality of this potential solution candidate is quantified by the evaluation process, which is used to guide the process to generate the next candidate. The creation of the new candidate occurs in the generation process as the cycle repeats itself.

The search process could be viewed as an intelligent trial and error approach to design. During each cycle of the search process, the guidance algorithm can use as little or as much information available from the representation and evaluation stages to make intelligent decisions about where to search next. There is a wide variety of methods to make an informed decision about which candidate will be generated next. These methods range from completely random, to using information about the past and/or the future (heuristic). If a candidate is evaluated to be a very good fit for a problem, the guidance process may choose to stop generating new candidates and alternatively return the best solution(s) found.

PipeSynth was made possible by the computational synthesis and design program known as GraphSynth (20). GraphSynth is a visual graph-theory based toolkit developed by Dr. Matthew I. Campbell and the members of Automated Design Lab at the University of Texas at Austin. PipeSynth is one of many automated design and optimization plugins developed for GraphSynth. These plugins are application specific additions to GraphSynth that demonstrate how a wide variety of design topics can be represented and solved with computational design synthesis.

2.3 Optimization Methods

During the guidance stage of computational synthesis, the optimization method attempts to decide the best way to try to find a better solution. Of the many different methods developed over the last few decades, each one has a trade-off between efficiency (speed) and robustness. More robust methods are more consistent at finding the best solution but at the cost of speed (21). On the other hand, more efficient methods can become tempting for more complicated problems due to their high convergence rate, but greater care must be taken to ensure that they will reliably return good results for a given application. Understanding and properly designing a problem is crucial in choosing an optimization method, and using it to its full potential.

2.3.1 Topological Tree Search Methods

Graph topology optimization methods are used to solve problems where the solution depends on a set of choices rather than a set of continuous variables. An example topology tree is shown in Figure 2.

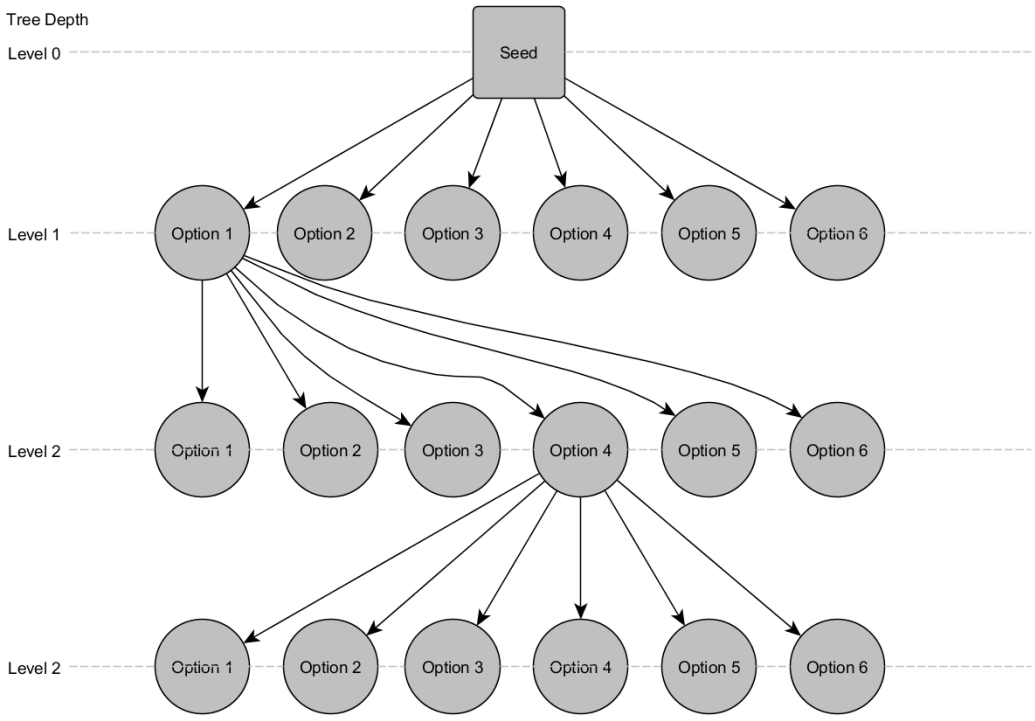

Figure 2. Example topology generation with consumable options

This figure shows how different choices can lead down a variety of paths and often return different results. Starting with the user represented problem (seed), all the possible options are generated and arranged like branches on a tree. At each of these new branches, there will be a new set of branches (options) to consider. If each option can only be used once, the partially expanded tree would look similar to Figure 2. Therefore, each time a branch is traveled to a lower level, the new options do not contain any of the previously used options in the path.

Topological search methods dictate how the exploration process is carried out. The two most fundamental methods that form the basis for many other tree searching algorithms are

breadth first search (BFS) and depth-first search (DFS) (21). Breath-first-search works horizontally across the tree by completely searching each level of candidates one at a time. On the other hand, depth-first-search moves straight down one branch of the tree by successively applying options until the bottom is reached. If the method hits the bottom of the tree, then it moves back up until there is another unexplored branch it can travel down. This process repeats until a satisfactory solution (goal) is found, or the entire list of possible candidates on the tree has been searched. Many problems, however, have no finite length to their branches because there will always be more options that can be applied. PipeSynth follows this pattern because there is an infinite amount of pipes and fittings that could potentially be chained together. Both BFS and DFS are known as uniformed searches because they search in a strict pattern with no intelligence of their own.

Informed methods, such as uniform cost search, use information about the tree to determine where to search at each iteration. These methods expand the tree based on a transition cost. During the generation of each child candidate, there will be some sort of cost, *g(n),* (not necessarily monetary) that is detrimental to the overall goal (objective). Therefore, the candidates with the lowest *g(n)* will be explored first. Heuristic methods, such as greedy search, try to pick a choice that will get them to the goal or solution as quick as possible, based on estimation of the cost to get to the goal, *h(n)*. A common combination of these two methods, *g(n)+h(n)*, is known as A-Star (A*). Each of these methods is application specific and highly dependent on the manner in which the problems are presented and the kind of information available to the search process(21).

2.3.2 Parametric Optimization Methods

Parametric methods are designed to find the best solution by finding the optimal values for a set of independent design parameters. The quality of each solution is measured by the evaluated values of a problem's objective functions. If other design criteria must be met other than the primary objectives, then the problem is said to be constrained.

If an objective function is analytical, then it may be possible to calculate the gradient. If the gradient cannot be readily calculated, it can also be approximated using finite difference methods. The Steepest Descent and Fletcher Reeves methods are two simple gradient-based

optimization techniques, but cannot directly handle constraints. Two popular complex gradient-based search methods that can handle constraints are Sequential Quadratic Programming and Generalized Reduced-Gradient.

Many optimization methods do not directly handle constraints due to their design, therefore constraints must be maintained by other means. A popular and relatively effective technique is to use penalty functions. For example, exterior penalty functions add to the resulting objective function whenever a constraint is violated. This way not only is the objective function being minimized, but the penalties from violated constraints are being minimized as well. Therefore, even if an optimization function is moving towards the true optimum, if it encounters an increasing penalty value, the optimization method will ideally tend to move back towards the best objective function evaluation value containing no constraint penalties. The problem with exterior penalties is the quality of the solution may become very sensitive to the magnitude of these penalties. Sometimes, to find a better solution the optimization method needs to transverse an infeasible region, but if the penalties are too high, it will never make it. Alternatively if these penalty weights are too low relative to the objective function, it may settle for a solution that is infeasible.

If a gradient is too computationally-intensive or impossible to determine, non-gradient search methods most be used. Common unconstrained, non-gradient based methods include: Cyclic Coordinate Search, the Nelder-Mead simplex method, Exhaustive Search, Random Hill Climbing, Powell's Search, Genetic Algorithms, and Simulated Annealing. The majority of these methods work towards the optimum by searching along a single direction in the n-design parameters space design at a time. The simplest, Cyclic Coordinate Search, searches the direction parallel to each design variable's unit vector, one at a time. In order to find a minimum along these directions, a line search method must be used. These one dimensional numerical methods seek to find the minimum function evaluation along a given line. Popular line search methods are: Golden Section, DSC-Arithmetic-mean, and DSC-Powell. Given a wide assortment of optimization methods, selection of the appropriate method has the potential to produce great results.

The object-oriented programming form of the representation and evaluation of networks in PipeSynth makes the direct calculation of an objective function's gradient difficult.

Having design constraints would favor optimization methods with active constraints, but methods such as SQP and GRG both require the gradient of the objective function. Finite difference approximations of the gradient can enable the use of these methods, but were ruled out early because of the high computational expensive of each network evaluation. Genetic algorithms and simulated annealing were deemed too stochastic at the parametric level. Therefore, this research focused on the following unconstrained, non-gradient based techniques: Nelder-Mead's Simplex Method, Cyclic Coordinate Search, Powell's Search Method, and Random Hill-Climbing.

3 Problem Analysis and Representation

PipeSynth seeks to produce the least expensive and least restrictive networks possible for each application. These two goals are known as the minimization objectives because they would ideally be a small as possible. However, the objectives must be met while producing feasible designs that take into account the constraints imposed on them.

3.1 Problem Definition

The optimization problem takes the form of minimizing two objectives subjected to three constraints.

$$\begin{aligned} &\textit{minimize} && f_{Cost}(\vec{x}),\ f_{Head\ loss}(\vec{x}) && && \textit{(1a)} \\ &\textit{subject to} && g_{PortsConnected}(\vec{x}) && \leq 0 && \textit{(1b)} \\ & && g_{PipeIntersection}(\vec{x}) && \leq 0 && \textit{(1c)} \\ & && g_{ObstacleIntersection}(\vec{x}) && \leq 0 && \textit{(1d)} \end{aligned}$$

The design variables vector, $\vec{x}$, is composed of both properties of a given topology and the parametric variables that orient and size the components of the topology. The cost objective, $f_{\mathrm{Cost}}(\vec{x})$, is the result of individual costs associated with both the topology and parametric design variables. The second objective, $f_{\mathrm{Head\,loss}}(\vec{x})$, minimizes the total head loss as a form of the total pressure drop across all the input and outlet ports. These objective functions are defined in detail, along with their evaluation, in section 4.1.

The three inequality constraints are all spatial constraints in order to prevent the optimization functions from trying to solve the networks with physically impossible designs. The first one, $\mathrm{g_{PortsConnected}}(\vec{x})$, forces the optimization methods to try to produce networks that have an open pipe within some specified tolerance of each output location. Every network begins synthesis at the input port locations, therefore there is no need for a constraint to join the network to these ports. The second constraint, $\mathrm{g_{PipeIntersection}}(\vec{x})$, prevents colliding pipes by specifying a clearance distance that they must maintain between each other. The third constraint, $\mathrm{g_{ObsaclestIntersection}}$, prevents components of the network from intersecting with any objects and any defined virtual boundaries residing in the world around the network.

One of PipeSynth's unique features is its integration of real pipe fittings and their associated geometric constraints. The use of commercially available fittings becomes more important as the number of fittings used per length of straight pipe increases. Alternatively, a pipeline that is comprised of miles of straight pipe will have such a large percentage of its cost associated with the straight pipe, that the cost of joining the pipes becomes relatively insignificant. However, for networks with multiple fittings in more confined areas, the fittings become much more significant to the overall design of the network.

Routing pipes to transfer fluid between objects typically requires that of the locations of the components that need to be joined are pre-determined. Whether this is a pump, tank, reservoir, heat exchanger, or nozzle, the pipe network is built to interconnect those much more significant and costly components. Therefore, the locations of these ports of these predefined objects define the scope and scale of the network. These ports also define the known fluid flow and pressure conditions for the network, along with the key fluid material properties.

PipeSynth is designed to correctly evaluate a large variety of common fluid flows. The various assumptions that encompass these fluids include the fluid to be a Newtonian fluid and to have constant density and viscosity over their working conditions. The flow is also assumed to be moving at a steady rate. Most liquids are considered to be incompressible at their working temperature ranges, with the most common example being water. Ideal or perfect gases can also be safely assumed to behave the same as incompressible flows, in terms of viscous effects, up to speeds of 0.3Mach, where the Mach number is the speed of sound in that fluid (1). The most common example would be air. There should also be a minimal amount of heat transfer to or from the walls of the pipes due to a temperature gradient. Depending on the application, careful analysis may make it possible to show that the temperature gradient is still not significant in considering the design of an optimal pipe network from a fluid flow standpoint.

The automated design approach presented here strictly focuses on the method of routing pipes between the network ports and not the analysis of interaction between components. There are many commercial software tools that can perform this analysis, and hence we are not recreating those tasks. Instead, the focus is on a method of representation, evaluation and optimization that allows for the automated synthesis of ideal pipe networks.

3.2 Representation and Synthesis

Before any problems can be solved or evaluated as a potential solution, they must be represented in a form that contains all the information about the layout and properties of the network. Beyond being a data structure it must also be in a structure that can universally represent any network within the scope of this program.

Efficient data structures and minimization of variables are important to the performance of any algorithm. First, every pipe and pipe fitting has a starting point and an endpoint in three-dimensional space. These can be simply stored as values in Cartesian coordinates. Furthermore, the diameter and type of each fitting must be identifiable. The representation and evaluation were primarily written specifically for this project in a total of approximately 2500 lines of C#. Features that are dependent upon only the candidate's topology that are also dependent upon the parametric design variables are separated from each other. Separating these features and selectively running these individual network properties generation algorithms only as needed is crucial to the performance of PipeSynth.

There are no analytical equations used to directly calculate value of each of the objective and constraint functions from the design parameters. These values are calculated from the resulting states and properties of the networks that are generated with each network design. For example, a new topology is generated with additional fittings. A sub routine determines which fittings belong to each channel and save them to a corresponding data structure. This routine does not need to be executed again until a new topology is analyzed. For each topology, the parametric optimization method determines a new set of design parameters. A sub routine then calculates the location of each fitting in three-dimensional space and flow states at each port of each fitting. This needs to be performed each iteration of the parametric optimization as it is dependent on the design parameters. Then the objectives and constraints are evaluated using these generated properties and conditions. This is more efficient than directly calculating each objective and constraint from the design parameters because many of the generated properties are used for multiple evaluations—for example, the Cartesian coordinates of each fitting.

3.2.1 GRAPH THEORY MODELING

Graph theory is the study of using graphs to model and represent the relationships and interactions between objects in a particular collection. Each object is represented as a vertex or node and each pair of interacting nodes is represented by a joining arc. A simple example graph with seven nodes and six directed arcs is shown in Figure 3 below.

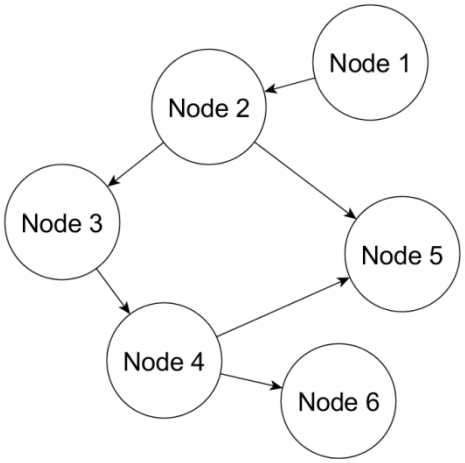

Figure 3. Example graph theory representation

The physical nature of branched fluid networks draws parallels to graph theory because each pipe fitting is paired to the next fitting through the use of straight pipes. Therefore, PipeSynth is represented in graph theory by setting the fittings as the nodes, and the interconnecting straight pipes as the arcs. Fluid flows between each node through each arc, starting from an input node and working its way towards the network outlets nodes. Every node in GraphSynth also has a position in three-dimensional space, and along with each arc, contains a data structure to retain information at each specific object. This allows information about the pipes and the problem to be represented in the graph.

The input and output locations are nodes that define the scope of the network. They each store a list (set) of strings called *locallabels* that allows them to be differentiated form other types of nodes. Additionally, each input has a direction vector that defines which way a pipe can be joined based on the applications of the existing network constraints or pipe orientations. If no direction vector is specified, the direction is defaulted to be parallel to the positive x coordinate of the seed graph. If a problem contains multiple separate channels that

must be solved simultaneously, then each network has a specific channel label that is applied to each node the in order for PipeSynth to differentiate them.

Each node's location is defined by one point in three-dimensional space corresponding to the intersection of the centerline axis of the ports of each fitting. However, each pipe fitting not only has a location, but also has an orientation in three-dimensional space that determines the directions of each of its ports. Rather than storing the location of the center of each port in every fitting, or representing the orientation with direction vectors, the location and orientation is determined with only two parameters. This is not only convenient, but necessary because if the pipe fittings were free to be moved and rotated in space independently of each other, then the fittings would rarely be oriented correctly to fit a straight pipe between them. Therefore, the location of each fitting is determined relative to the previous fitting, starting at the given input node. A fitting's location is determined by the length of pipe between it and the previous fitting. These corresponding lengths and angle of attachments are represented by the parameters α and Θ respectively. Figure 4 shows how simple topology comprised of one 90 degree elbow and two straight pipes is updated in three-dimensional space based on the design variables.

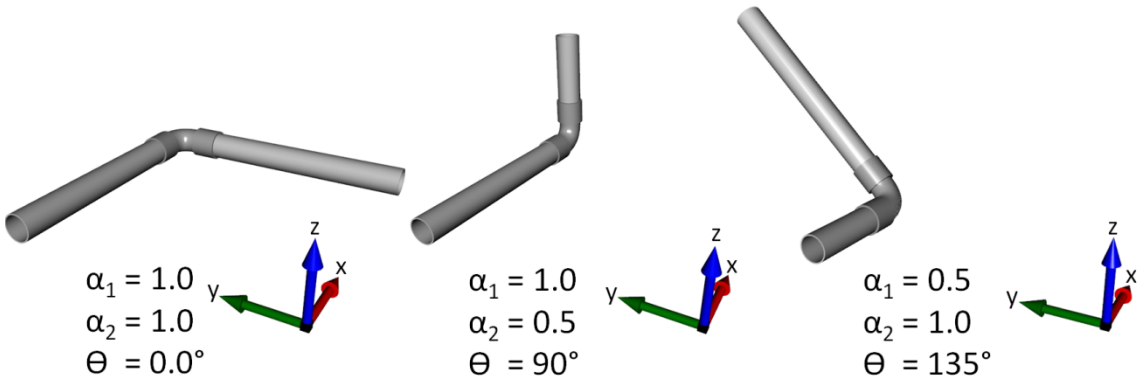

Figure 4. One topology generates different networks based on design variables

In Figure 4 variables α_1 and α_2 scale the lengths of the first and second pipes respectively. The variable Θ, ranges from zero to 360 degrees and rotates the elbow fitting about its input port (each fitting has exactly one defined input port). Additional fittings and pipes follow this same generation methodology.

This representation of the network with these design parameters serves two important purposes. First this system reduces the number of design variables significantly compared to representing the x, y, and z coordinates for every node as independent design variables. Second, this system guarantees each pipe will be attached at the correct geometric relationship with the previous fitting. If the Cartesian coordinates where allowed to be freely varied as independent parameters, then there would need to be another constraint to enforce that the fittings align correctly.

The diagram in Figure 4 is specific for the generation of an new straight pipe that joins a fitting with perpendicular ports. Non-orthogonal port fitting types, such as 45° elbows, follow this same system except they need one additional step. After the orthogonal placement is determined as shown in Figure 4, the new straight pipe must also be rotated appropriately about the center of the fitting, for example 45° outwards, in order for the endpoint of the second straight pipe to generated correctly**.**

3.2.2 SEEDS

In generative grammar, the seed is the starting graph that represents a particular design problem. Sometimes, this can be very abstract, but for PipeSynth it is fairly representative of the real world problem. Each problem, or seed, is fundamentally defined by location of the input and outlet ports in Cartesian coordinates. These ports are represented by nodes in GraphSynth and contain pertinent data about their particular port such as predefined pressure, flowrate, and the fluid channel they belong to. Inlet ports are also used to define key fluid material properties— most importantly density and viscosity.

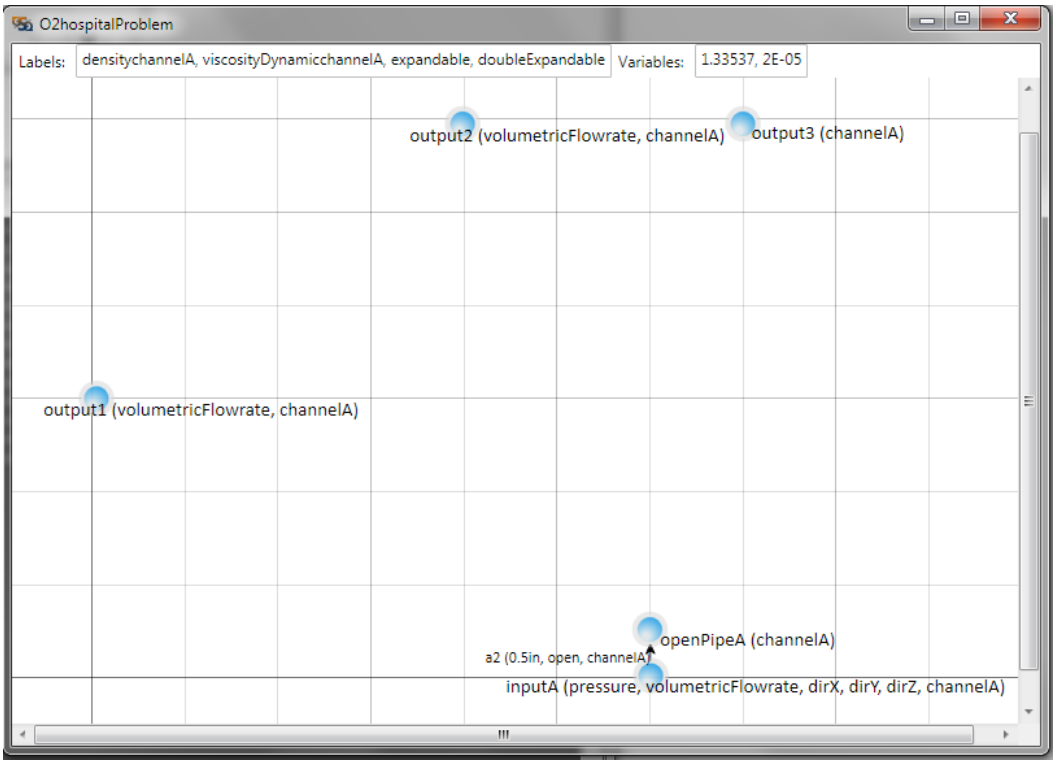

Figure 5. Example seed for PipeSynth with one fluid channel with one input and three outputs

Branched networks in PipeSynth are created by starting at one inlet and working towards all the remaining input and outlet ports. For consistency and performance, if more than one input is required per channel, then all the additional inputs are represented as outputs but with negative flowrates. PipeSynth normally assumes positive flowrates at output to be flowing out of the system and using this technique to represent multiple inputs results in the same fluid dynamics states as those that would result by having multiple inputs nodes.

3.2.3 Grammar Rules

One of the GraphSynth's primary features is the creation and application of grammar rules to conditionally manipulate and generate new graphs. PipeSynth uses grammar rules to construct the topology by adding one applicable pipe fitting at a time. Each grammar rule in PipeSynth consists of three or more nodes that represent the inlet port, the fitting origin and type, and the outlet port. An example of a half-inch 90° nominal pipe size, NPS, elbow fitting is shown in Figure 6 below.

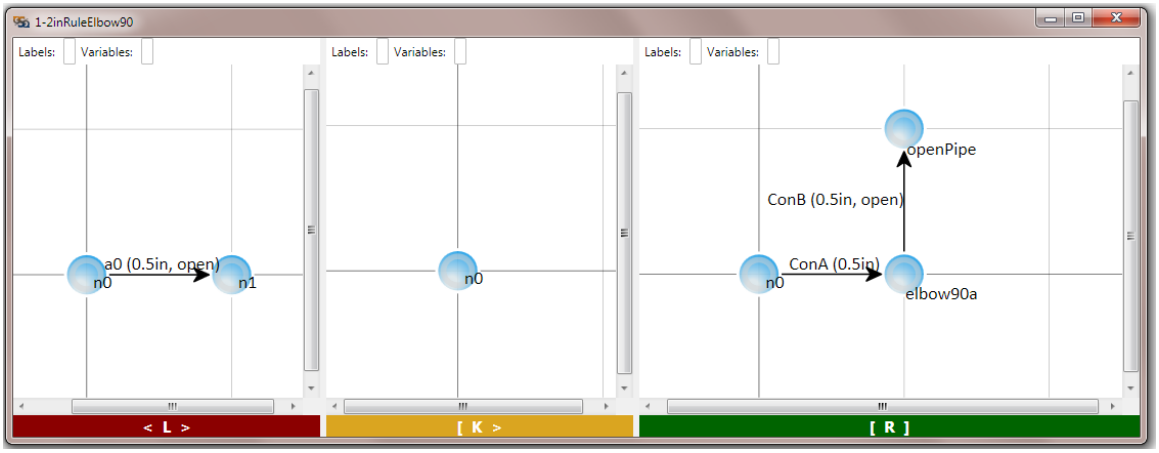

Figure 6. PipeSynth grammar rule for a half inch NPS 90° elbow fitting

Each grammar rule in GraphSynth is defined by three sub-graphs: L, K, and R. The left graph, L, is the sub-set of node and arc relationships that must be recognized by GraphSynth in order for the grammar rules to be applied. In this example, there are two arbitrary nodes with an arc joining them containing the labels *0.5in* and *open*. The open label appears only in PipeSynth on arcs directed out of unconnected ports. The *0.5in* label specifies the diameter of the recognized port must be a half inch, the same size as inlet port on the fitting that will be applied with this rule. Therefore, this rule can only be applied to unconnected ports of fittings, with an open label, that have a NPS of 0.5 inches.

The middle graph, K, represents the nodes and arcs kept from the L graph to be reused in the R graph. For all grammar rules in PipeSynth, this is the existing pipe fitting that the grammar rule will build on.

The right graph, R, represents what the final nodes and arcs will be after applying the graph. For the example, in Figure 6, the R graph contains the old fitting node, *n0*, the new fitting node, *elbow90a*, and two arbitrarily named arcs with half inch labels. The second arc, *ConB*, has the label open, in order for another rule to recognize that this is now an open port for the fitting *elbow90*. The extra node, *openPipe*, has no physical representation but is required in GraphSynth to create the arc *ConB*, because each arc is required to join two arcs. Each grammar rule in PipeSynth contains at least one of these new open pipe node-arc pairs to allow for the future expansion of this branch of the network.

In order to expand a network, the grammar rules recognize where they can be applied by first finding a node and arc pair that represents an open, or free, straight pipe. Then it compares local variables of the open pipe arc and the inlet pipe arc to determine if they are equivalent sizes. If a match is found, then it is considered to be a possible option in building onto the current candidate.

When an option or grammar rule is applied, it removes the open node and arc pipe pair and replaces them with the new nodes and arcs of the grammar rule. Examples of the rule in Figure 6 being applied to a simple seed in Figure 7 is shown in Figure 8.

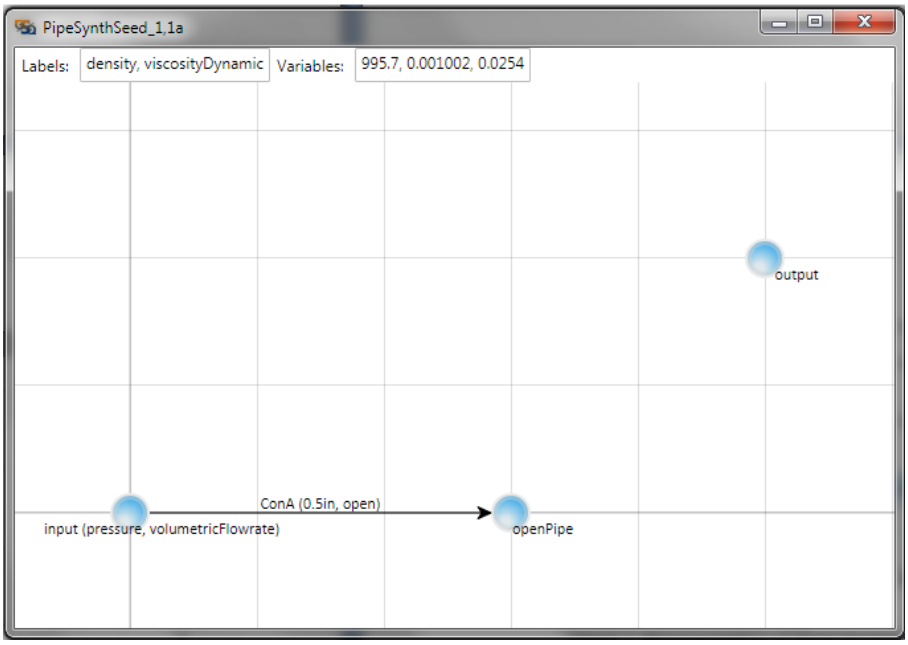

Figure 7. Example of a seed before rule application

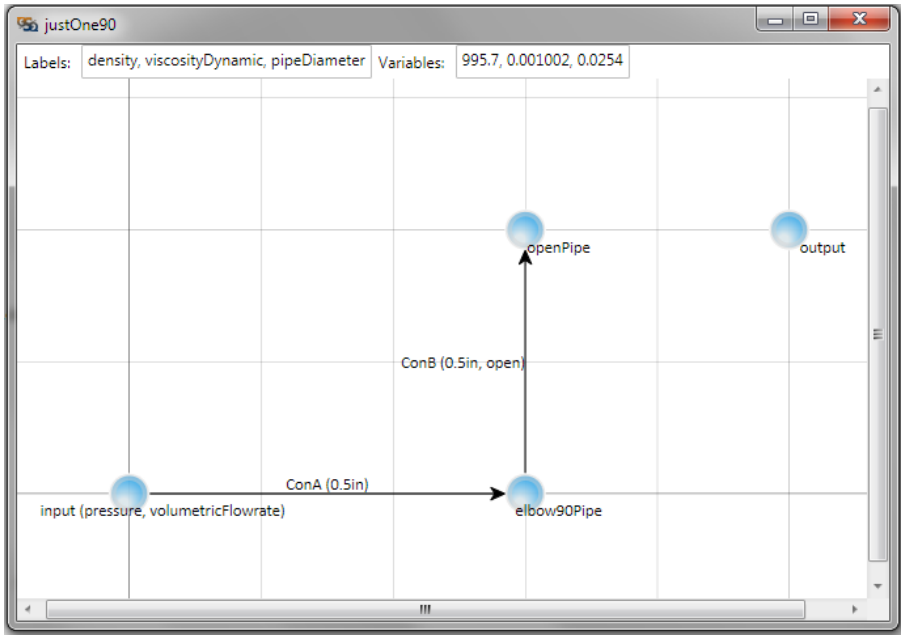

Figure 8. Example graph after grammar rule application shown in Figure 6

4 OBJECTIVES

The evaluation value of the objectives is determined by both the topology and the design parameters determined by the parametric optimization function. In order to properly evaluate each of the objectives and constraints in PipeSynth, further insight and understanding is needed for each topic. Understanding the theory needed to solve these functions not only adds confidence to validating each one, but helps in producing more efficient algorithms.

This section discusses how the objectives and constraints were created and the theory that was used to develop the C# algorithms such that they can be evaluated for use during the optimization. Due to the multi-objective nature of this piping network an approach to controlling this conflicting problem is discussed in section 6.3 with Pareto optimality.

4.1 COST EVALUATION

The first objective, $f_{\mathrm{Cost}}(\vec{x})$, seeks to minimize the total construction cost of network and its components. The total costs is comprised of several types of costs. First, installation and construction costs are a combination of material, assembly, and bracketing costs. The calculation of material cost comes from the number of fittings and linear straight pipes. Each component's cost is based on the prices from the list of available fittings for each application. Assembly cost can be a little more difficult to estimate, but could be approximated by first determining the assembly costs for attaching a pipe to every type and size of fitting. This attachment cost would be based on the joint type of the fitting, including: welded, bonded, compressed, threaded, bolted, etc. Then the total assembly cost would be the sum of these joint costs for every port in every fitting including the transportation and installation costs based on the average length of the straight pipes. Bracketing, or structural support, costs are application specific and are not incorporated into PipeSynth's objective function at this time. One additional cost to consider is the networks operating costs. These are also not approximated by PipeSynth, but would be included in the maintenance and a total life cycle costs.

Cost is calculated primarily as summation of individual component costs. In addition, these costs are dependent both on the topology and design variables. The topology determines how many of each specific fittings and straight pipes there are. Each fitting has a unique price

depending on its material, specification and size (port diameters). Each straight pipe has a material cost per liner foot depending on similar conditions. This length is determined by the α design variables that are determined during the parametric optimization stage. The total cost can be viewed as a summation of the topology's costs and the costs dependent on parametric design variables, equation 2.

$$minimize\ f_{cost,total} = f_{cost,topology} + f_{cost,parametric} \quad (2)$$

The topology cost is the summation cost of each fitting cost, and assembly cost of each fitting. This is display below in equation 3.

$$f_{cost,topology} = \sum_{i=1}^{m} \left(f_{cost,material,fitting}(fitting_i) + f_{cost,Assembly}(fitting_i) \right) \quad (3)$$

If m is the total number of fitting nodes in a specific topology, then each fitting gets passed onto the material and assembly cost sub-functions. The material costs function simply looks up the part cost based on size and material of the fitting passed into the function. The default prices in PipeSynth are for schedule 40 PVC pipe fittings as listed at the McMaster-Carr® company(22). PipeSynth was designed to accept a wide variety of pipe types and sizes and new prices and specifications can readily be added into the program as required. The assembly cost is based on the assembly cost of each port and the associated set assembly cost for that fitting class and diameter.

The parametric cost determines the cost of the straight pipes used to join the fittings contained in the topology. After a specific topology has been optimized parametrically, the lengths of each pipe in a candidate are known. Therefore, the candidate is passed into the parametric cost function, equation 4.

$$f_{cost,Parametric}(topology) = \sum_{i=1}^{m} (pipe_i.lenght * linearPipeCost_i) \quad (4)$$

This function sums all straight pipe material costs by multiplying each individual length by its respective linear cost. After all the individual costs are found, the total cost, equation 2, is returned.

4.2 HEAD LOSS EVALUATION

The second objective, $f_{\text{Head loss}}(\vec{x})$, goal is to minimize the resistance of the network. This can be viewed in terms of pressure drop across the different ports. The network resistance is measured by the second objective by calculating the total pressure drop, Δp_i, across each inlet and outlet ports for every channel. This forms the second objective, $f_{Head\ loss}(\vec{x})$, shown in equation 5 below.

$$\textit{minimize}\ f_{Head\ loss}(\vec{x}) = \sum\left(\Delta p_i(\vec{x})\right) \tag{5}$$

This minimization of losses is a good way to measure the efficiency of a network design, a very common motivation in design optimization.

The fluid dynamics determine how efficient or resistive a network is to a fluid transversing through it. Therefore, proper understanding and accurate approximations of the behavior of the fluid within the network is crucial in finding the best network design solutions. Starting from the ports, each one will be either be a source or sink of mass and energy for the network. If the fluid flowrate and pressure are nearly constant with time, then the system is said to have steady-flow. Between these ports, however, the pressure and flowrate will vary. In order to properly determine the pressure drop, the state of several intermediate points must be calculated. The state of every fluid element is governed by three primary relations: the conservation of mass, the conservations of momentum, and the conservation of energy.

The conservation of mass principle states that for any fluid element, mass cannot be created or destroyed. If the flow can be assumed to behave as an incompressible fluid (valid for most fluids and ideal gasses moving at a steady rate of less than 0.3Ma), then the density is constant and the relation simplifies to equation 6 for any fixed volume, V.

$$m = \rho * V = const \tag{6}$$

If the net mass of the system is constant, then the rate of change of mass with respect to time (mass flowrate) must be equal to zero. This leads to the conservation of mass flowrate across all the network's ports, equation 7.

$$\Delta \dot{m}_{system} = 0 = \sum_{i=1}^{n_{ports}} \dot{m}_{port,i} \tag{7}$$

The term $\dot{m}_{port,i}$ indicates the first time derivative of mass that passes the cross-sectional area of a port i, while the term n_{ports} equals the total number of ports in the system. Substituting equation 6 into equation 7, and dividing by density, leads to equation 8.

$$\Delta\dot{V}_{system} = 0 = \sum_{i=1}^{n_{ports}} \dot{V}_{port,i} \quad (8)$$

This equation is the conservation of volumetric flowrate and is a tool for solving fluid networks. It can be applied to entire network or any sub-section, including individual fittings, to aid in calculation of unknown flowrates. The inlet ports are defined as a point of adding mass to the system. Therefore outlets in the networks must have negative flowrates to conserve the mass of the system. If all but one flowrate is known for the ports of a network, then equation 8 can be used to analytically solve for the final port flowrate.

Conservation of momentum is similar in principle to conservation of mass except that momentum can be lost inside the system due to viscous friction. The principle states that the net rate of change of momentum of the system must be zero, or equivalently the sum of the forces must be zero. In pressure driven pipe flows, the rate of momentum being added to the system at a port is equal to the mass flowrate multiplied by the mean velocity of the fluid passing through that cross-section. The direction of the flow determines the sign of the mean velocity and therefore, if the rate of momentum is added to or lost from the system. With no net change in the rate of momentum, the sum of the rate of momentum added or removed at each port must be balanced by the momentum loss at the walls through shear stresses, τ_w, induces by viscous effects.

$$\Delta \frac{d(mU)}{dt}_{system} = 0 = \sum_{i=0}^{n_{ports}} \dot{m}\bar{u} - \int_{A_s} \tau_w dA \quad (9)$$

The term U represents the instantaneous velocity vector at a given point, and $\bar{u}$ is the mean velocity found by dividing the volumetric flowrate by the cross-sectional area that the fluid is passing through (equation 10).

$$\bar{u} = \frac{\dot{V}}{A_C} \quad (10)$$

This mean flowrate is shown compared to the actual parabolic velocity profile typical of viscous incompressible fluid flows in Figure 9.

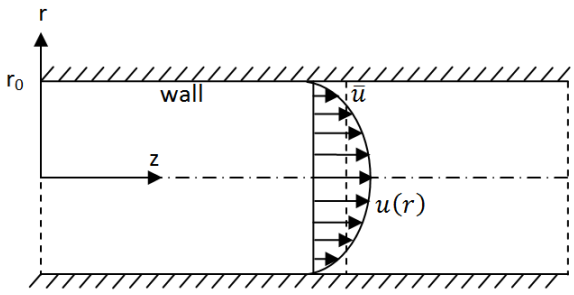

Figure 9. Typical parabolic velocity profile for viscous incompressible pipe flows

The parabolic profile results from viscous fluids matching the velocity of interacting solid surfaces—in this case the inside pipe wall. This assumption is known as the no slip condition.

The local shear wall stress is equal to the derivative of the fluid's velocity with respect to the direction of the surface normal, evaluated at the surface wall (r_0), and multiplied by the fluid's viscosity. This is shown in equation 11 below (1).

$$\tau_w = \mu \left(-\frac{du(r)}{dr} \right)_{r=r_0} \tag{11}$$

The negative sign arrives as an artifact of the coordinate system being inverted, with r = 0 at the center of the pipe, causing the surface normal to be pointed in the negative radial direction. The term r0 is the internal radius of the pipe and indicates the location of the inside pipe surface of the wall. From a sum of forces point of view, this shear stress is equal to the force of the pipes reacting against the drag force of the fluid. The specific velocity profile, $u(r)$, of a pipe flow varies depending on the type of flow, but will be accounted for later in the coefficient of friction correlation that is also specific to each type of flow.

The most fundamental characterizing relationship in fluid dynamics is the nondimensional Reynolds number that is shown in equation 12.

$$Re_L = \frac{\rho \bar{u} L}{\mu} \tag{12}$$

Physically it represents a ratio of a fluid's momentous effects to its viscous effects, where ρ is its density, μ is its viscosity, $\bar{u}$ is its velocity, and *L* is characteristic length dependent on the flow type. For internal pipe flow, *L* equals *D*, the internal diameter of the pipe ($2r_0$). The Reynolds number is used for characterizing the behavior of fluid flow for a wide variety of applications and is also used as a key parameter in a many flow dependent correlations.

The characteristic types of fluid flow are broken into three primary regimes: laminar, transitional, and turbulent. Laminar flow is used to describe smooth flows that are usually slower moving and will be maintained up to a Reynolds number of about 2000 for internal pipe flows. Turbulent flows are chaotic and difficult to predict at a particle level. Local fluid velocities are very random and follow no repeating flow pattern. Fully turbulent pipe flows typically have a Reynolds number above 4000.

Transitional flow is the intermediate stage between laminar and turbulent flow where a fluid shows local traits of both laminar and turbulent flows. The rate of transition towards a fully turbulent flow is highly sensitive to disturbances in the flow and environment. This makes it hard to predict what percentage and which types of turbulence traits have developed for each particular turbulent flow. Therefore, transitional flow is very difficult to predict based on the Reynolds number alone.

Some special cases of laminar flows need to be treated differently to see if they still satisfy the no-slip at the wall assumption made earlier. There is a possibility of slip for micro fluid flows, and laminar gases either with a very low Reynolds numbers or a very high Mach numbers (1). However, these flows will not be discussed further due to their special treatment and small level of occurrence in the real world. For each flow regime, the velocity profile takes on a different shape and therefore a different resulting wall shear stress.

The velocity of profile, is not only dependent on the Reynolds number, but also on how far a fluid has traveled in a pipe after a large change of geometry. If the pipe was connected to a a large tank, the velocity profile would initially look very uniform across the radius of a pipe (slug flow). The velocity profile would then transition along the z axis towards becoming more parabolic until it eventually becomes fully developed and maintains a constant velocity profile. The distance downstream of a large change of geometry at which laminar flow can be considered fully developed is known as the entrance length and is shown in equation 13 (1).

$$L_e \approx 0.5D + 0.05 * Re_D * D \quad (13)$$

The entrance length for laminar flow is limited to about 100 diameters because Re_D is limited to about 2000 for laminar flow. Turbulent flow uses another correlation but with a similar purpose, shown in equation 14.

$$L_e \approx 4.4 * {Re_D}^{1/6} * D \quad (14)$$

PipeSynth assumes there is fully developed flow at each inlet port because the stage of development at each inlet port is very problem specific. Also, the effects of non-fully developed flow are small and usually have negligible contribution to the overall pressure drop of a network. The objective of PipeSynth is to find the best possible solution to pipe network problems and even if it overestimates the pressure drop slightly, it would be doing so for all the possible solutions and should still pick the best, or nearly the best solution.

To account for these different flow types and their corresponding velocity profiles, it is customary to nondimensionalize the wall shear stress by dynamic pressure to form the skin-friction coefficient. Known as the Fanning friction factor, C_f, it is shown in equation 15 below.

$$C_f = \frac{2\tau_w}{\rho \bar{u}^2} \quad (15)$$

This friction factor can then be determined based on a fluid's flow regime. It should be noted that another parameter known as Darcy friction factor, λ, is equal to four times the Fanny friction factor, C_f, and care should be taken when using different correlations that the proper friction factor is used. For laminar pipe flows, the equation for friction factor is widely accepted and backed by theoretical formula to be the simple ratio in equation 16.

$$C_f = \frac{16}{Re_D} \quad (16)$$

Transitional flow has no direct correlations to predict friction factors, however, Bhave notes that it is safe to assume the Darcy friction factor to be between 0.03 and 0.08 for water in the transitional regime (4). In order for PipeSynth to accommodate a wide variety of fluids, it conversely overestimates the friction factor using its fully turbulent correlation.

Turbulent flow, while chaotic, has very predictable mean flow when averaged over time. This has led to several correlations. One of the better correlations is an explicit form of the traditional implicit Colebrook correlation for the Darcy friction factor, and is shown in equation 17 below (4).

$$\frac{1}{\sqrt{f}} = -2log\left[\left(\frac{4.52}{Re_D}\right) log\left(\frac{Re_D}{7}\right) + \frac{e}{3.7D}\right] \quad (17)$$

The term Re_D is the Reynolds number having a characteristic length equal to the pipe's interior diameter and variable e is the surface roughness material property of the inside of the pipe. This correlation has an accuracy of ±0.30 percent for Reynolds numbers between 4000 to 10^8 and e/D from 0 to 0.05, which covers the majority of turbulent flows (4). These friction factors are all used to calculate what are known as the major head losses of networks, which occur in the straight pipes. The fittings and joints are the locations of minor head losses and, with their complex geometry, have much more complicated velocity profiles and friction factors. However, good correlations have been developed which scale major friction factors to equivalent minor head loss friction factors based on the individual fitting geometries. The minor head losses can be said to have the same resistance as a much longer straight pipe of the same type and diameter. For example a 90° elbow is said to cause about the same amount of friction as straight pipe 24 times its diameter long. Therefore, a one inch elbow would have the same resistance on the network as 24 inches of straight pipe. A table of various pipe fittings' equivalent length per diameter is shown in Table 1(4).

Table 1. Equivalent pipe lengths for various fittings (4)

	Equivalent Pipe Length per Diameter
Sudden Enlargement	
D1/D2 = 3:4	7
D1/D2 = 1:2	22
D1/D2 = 1:4	31
Sudden Contraction	
D2/D1= 2:1	12
Bends an Elbows	
Flanged 45°	6
Flanged 90°	24
Tee	
Line/Runner	60
Branch	300

With the friction factor determined from the appropriate correlation, the shear stress at the wall can be calculated directly as in equation 18.

$$\tau_w = \frac{C_f \rho \bar{u}^2}{2} \tag{18}$$

Converting from mean flowrate $\bar{u}$, to volumetric flowrate, $\dot{V}$, results in equation 19.

$$\tau_w = \frac{8 C_f \rho \dot{V}^2}{\pi^2 D^4} \tag{19}$$

Now, the full conservation of momentum equation can be solved for a fixed control volume. The general form is shown in equation 20, below.

$$F = \frac{d}{dt}(mU) = \frac{d}{dt}\int_{CV} U\rho \, d\, V + \int_{CS} U\,(\rho U \cdot dA) \tag{20}$$

Considering just one pipe of finite length, the total force acting on the pipe, F, through the control volume, the interior volume of the pipe, would be equal to the shear wall stress, τ_w, integrated over the pipe wall. However, the shear stress is constant for fully developed incompressible pipe flow over the length of a constant cross-section pipe. The flow is steady so the rate of change of momentum in the control volume, and therefore the third term of equation 20, will be zero. The final term represents the momentum entering and leaving the

ends of the pipes. If we assume a uniform pressure distribution across the pipe's cross-section at the inlet and outlet, then this simplifies down to being the pressure times the cross sectional area, taking care to note signage of the pressure acting on the control surfaces. With the integrations complete, the conservation of momentum equation becomes 21.

$$\tau_w * A_s = (P_2 - P_1) * A_c \quad (21)$$

Substituting the area calculations and rearranging in terms of pressure drop results in 22.

$$(P_1 - P_2) = \frac{-4\tau_w * \pi DL}{\pi D^2} \quad (22)$$

Substituting in the shear wall stress, equation 19, and simplifying results in the equation for pressure drop with a given flowrate, 23.

$$(P_1 - P_2) = \frac{-32 C_f \rho L \dot{V}^2}{\pi^2 D^5} \quad (23)$$

The objective function minimizes units of head loss. The relationship between pressure change and head loss is similar to change in potential energy and is found by equation 24 below.

$$h_l = -\frac{\Delta P_{1-2}}{\rho g} \quad (24)$$

In terms of pipe flow analysis, this head loss is also encountered whenever a pipe moves against gravity, whereas an increase in pressure occurs when flow moves with gravity. Therefore, it is important to note the change of pressure not only due to friction, but also due to gravity. By converting equation 23 to head loss, the resulting relation is the well published Dacry-Weisbach head loss equation, but scaled by a factor four (4). This is expected because the Fanny friction factor that is used here is equal to one fourth the scale of the Darcy friction factor.

$$h_l = \frac{32 C_f L \dot{V}^2}{\pi^2 g D^5} \quad (25)$$

By applying these pressure drop equations to sections of the network with one known pressure and a known flowrate, the alternate pressure can be directly solved for.

The principle of conservation of energy in typically used to determine the behavior heat transfer and thermodynamics. The total energy of the system, $\dot{\mathrm{E}}_{\mathrm{system}}$, is dependent on the heat into a system, $\dot{\mathrm{Q}}_{\mathrm{system}}$, and the work done by the system, $\dot{\mathrm{W}}_{\mathrm{system}}$. This is defined in equation 26.

$$\dot{E}_{system} = \dot{Q}_{system} - \dot{W}_{system} \tag{26}$$

If the heat rate into, or exiting, the system by heat transfer due to temperature gradients is determined to be very small, as is assumed at this point in time with PipeSynth, then the only change of energy is from mechanical to thermal due to the viscous effects. With the pressure drop formula already derived, there is no need to investigate the conservation of energy equation further.

Knowing the state of a fluid at every junction, or fitting, is a critical part of evaluating the total head loss across the ports. The pressure drop of a fluid from an input to an output is equal to the summation of pressure drops across any continuous path that connects them. Similar to using node voltage techniques in circuit theory, the network is divided into a system of isolated subsections. Each section has a constant flowrate, and a unique change of pressure due to viscous friction and gravity. These sections are represented by objects known internally to PipeSynth as Bernoulli objects. They are named after Daniel Bernoulli, because they relate the pressure change between two points by the change of kinetic and potential energy. The Bernoulli object is analogous to a resistor in circuit theory with each subnode having a specific "voltage" (Pressure) and "current" (Volumetric flowrate). However, in this case, flow resistance is not directly proportional to flowrate, as electrical resistance is to current. Therefore, circuit theory equations cannot be directly applied.

The fluid state at each Bernoulli object endpoint is not unique, but is shared with at least one other Bernoulli object, with the exception of the network port nodes. Thus, the state of the fluid at these shared points will be used more than once, and it is more computationally efficient if each Bernoulli object points to exactly two of these new types of objects. These additional non-physical objects are known as subnodes and are always associated with one node and one arc, and represent the fitting port at that junction. Figure 10 shows a diagram representing the location of each subnode and each connecting Bernoulli object.

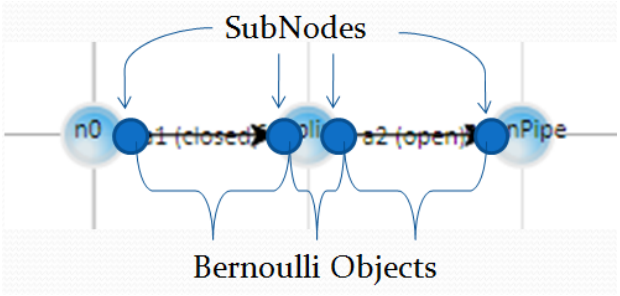

Figure 10. Representation of fluid dynamic states as Bernoulli and SubNode objects

Regular nodes are not used as the location of fluid states because, in addition to there being pressure drops in each straight pipe, there is also a unique pressure drop across each of the ports in each pipe fitting. For example, tee fittings have two collinear ports and one orthogonal port. The pressure drop by the fluid turning between any two orthogonal ports is much greater than the pressure drop between the two collinear ports. Therefore, there will be a unique pressure at each port of each fitting, and it is necessary to have the fluid state represented at each of those locations by a subnode object. By iteratively solving the fluid mechanics equations for each Bernoulli object, the state of each of these subnodes and the mean state of the fluid at any point in the entire network can be determined.

To find the states of the fluid at every subnode in the network, variations of equations 8 and 23 are used to solve each Bernoulli object iteratively until the entire network is solved. This object-oriented approach is intelligent in that is solves for the easiest states first, working towards more difficult. For example, if one flowrate is known at one subnode in a single input, single output, SISO, pipe or fitting, then the opposite flowrate must be the same because of equation 8 and the incompressibility assumption. If a fitting has multiple inlets and/or outlets, and only one flowrate is unknown, then it can be solved with equation 8 as well. This same reasoning also applies if there is only one unknown flowrate for the network ports of a particular channel.

A slightly more difficult condition is when the flowrate of a Bernoulli object is known but only one pressure is known. Then equation 23, can analytically solve for the unknown pressure after choosing the proper friction factor correlation based on the fluid's Reynolds number. Using these techniques alone, no numerical methods are needed to approximate any fluid

states as long as one pressure is given along with at least one less than the number of network ports flowrates is given. The developed algorithm for performing this fluid mechanics simulation is written in C# and has 592 lines of code.

5 CONSTRAINTS

The constraints are key to finding and producing feasible solutions. In PipeSynth, all three constraints are inequalities because they are limiting or preventative constraints. Inequalities let the optimization methods know if, and by how much a constraint is being violated. The greater the magnitude of that violation, the greater the penalty on the objective function for that solution. The magnitude of a feasible solution for all of PipeSynth's is irrelevant as it provides no benefit to a well defined problem. Therefore, the return value of any feasible constraint in PipeSynth will always be zero, and the return value of any infeasible constraint will be positive.

In order to test if network designs are possible to build in the real world, they must be tested for intersections with other objects and boundaries to make sure that no objects occupy the same space. Full three-dimensional solid object intersections can be complicated and unnecessarily resource intensive. To avoid this problem, the pipes are converted into parametric line segments and the minimum distance between each object and each line segment is calculated. Some tricks are employed to improve the efficiency of these tests.

First, all line segments are represented parametrically, as in equation 27.

$$\vec{P}(t) = \vec{P}_1 + t(\vec{P}_2 - \vec{P}_1) \tag{27}$$

Therefore, any point on a line, $\vec{P}(t)$, can be defined by one parameter, t, which is only between zero and one. Then if the vector starting from $\vec{P}_1$and pointing to $\vec{P}_2$ is labeled $\vec{P}_{1,2}$, a simple calculation yields the closet distance, d, between the line and a point of interest, $\vec{P}_x$.

$$d = \frac{|(\vec{P}_{1,2}) \times (\vec{P}_x - \vec{P}_1)|}{\|\vec{P}_{1,2}\|} \tag{28}$$

The intersection of a line segment and an infinite plane equation can be found by knowing the normal to the plane, $\vec{N}$, and any point $\vec{P}_0$ that lies in the plane as in equation 29.

$$t = \frac{-\vec{N}\cdot(\vec{P}_1 - \vec{P}_0)}{\vec{N}\cdot\vec{P}_{1,2}} \quad (29)$$

If this equation yields a value between 0 and 1, then there is an intersection with the line segment. To find this location, this value of t is entered into parametric equation of a line (equation 27). By using a few vector geometric calculations, the intersections of objects in PipeSynth can quickly and efficiently be found.

5.1 Port Connections Evaluations

The first constraint, $g_{PortsConnected}(\vec{x})$, is a way to test if the output ports have been reached by a network design. Specifically, this constraint tests if each design output location resides within some tolerance distance of the desired output location.

$$g_{PortsConnected}(\vec{x}) = \sum\big(max\,(d_{output,i}(\vec{x}) - \ tolerance, 0)\big) \leq 0 \quad (30)$$

The function $d_{output,i}(\vec{x})$ is the distance between a corresponding output port and its paired open pipe. As long is $d_{output,i}(\vec{x})$ is less than the specified tolerance, then that output will not contribute to anything in the summation because the *max(x,0)* function will return 0. The *max(x,0)* function is important because it keeps each designed output location test independent of the next. If the outputs are not within in the tolerance $g_{PortsConnected}(\vec{x})$ becomes more positive the further away they get.

The port connection constraint will always seek to minimize the distance between each open pipe and the nearest output location (that both belong to the same channel). This is the only constraint that is allowed to be within some tolerance value and still considered feasible. This arrives from the unnecessary precision of floating-point numbers that prevent this distance from ever being truly zero. The distance between each pair of nodes is simply the magnitude of the vector that joins the two. For each of these distances that are above the specified tolerance, a penalty proportional to that distance is added to return value of this constraint. Otherwise if the nodes are within the tolerable distance of each other, no penalty is contributed by that pair.

5.2 Pipe Intersection Evaluations

The second constraint, $g_{PipeIntersection}(\vec{x})$, prevents the pipes from colliding with each other.

$$g_{PipeIntersection}(\vec{x}) = \sum_{i=0}^{n_{pipes}} \left[\sum_{j=i+1}^{n_{pipes}} \left(PIT\left(pipe_i, pipe_j\right) \right) \right] \leq 0 \qquad (31)$$

The function $PIT\left(pipe_i, pipe_j\right)$ evaluates two specific pipes and returns the results of a pipe intersection test on them. The constraint's value increases by an amount proportional to intersection magnitude of each pair of pipes. For each pair of pipes that don't intersect, the pipe intersection test will return zero. The summations starting indices ensures that every pair of pipes, up to the total number of pipes, n_{pipes}, are tested exactly once.

If the goal is just to prevent collisions from happening, then it does not matter where the intersection occurs, only if and by how much. With this consideration, the minimum distance between the central axes of the two pipes will tell whether or not an intersection will occur. To account for the fittings that will be larger than the diameter pipes and have more complex geometry, the pipes are approximated as cylinders that run all the way beyond the center point of the fittings, which occurs at the intersection between the centerlines of all pipes connected to a particular fitting. Then by adding a large enough clearance value to outer diameter of the pipes, adequate clearance can be assured for all the pipes and fittings as long as $g_{PipeIntersection}(\vec{x}) \leq 0$. Equation 28 was presented to test the minimum distance between two infinite lines. However, the pipes each have finite length, therefore the minimum distance could occur at one of three conditions (23).

First, the shortest distance could occur between two of the pipes' endpoints. Second, it could also occur between the endpoints of one pipe axis and some unknown internal point on the line segment representing the second pipe. Finally, it could occur between two unknown internal points along the pipes' two axes. PipeSynth first finds out which of these three conditions the relation of two pipes exhibits, then finds the minimum distance either between them as two endpoints, an endpoint and an infinite line, or two infinite lines. If this distance is greater than the sum of the two external radii of the pipes, plus some clearance value, then the constraint returns no penalty. Otherwise it returns a value proportional to violation distance.

5.3 Obstacles and Boundaries Intersection Evaluation

The third constraint, $g_{ObstacleIntersection}(\vec{x})$, prevents any part of the network from residing in the same physical space as real world objects. The obstacle constraint function guides the pipes to stay within predefined areas by calculating a magnitude of intersection violation, similar to the pipe intersection test.

$$g_{ObstacleIntersection}(\vec{x}) = \sum_{i=0}^{n_{pipes}} \left[\sum_{j=i+1}^{n_{objects}} \left(OIT\left(pipe_i, object_j\right) \right) \right] \leq 0 \qquad (32)$$

The function $OIT\left(pipe_i, object_j\right)$ is the object intersection test that checks for intersections between each pipe and each defined object/boundary. Similar to $g_{PipeIntersection}(\vec{x})$, the value of $g_{ObstacleIntersection}(\vec{x})$ is proportional to the sum of the magnitudes of the intersections for each pipe.

Obstacles are designed to represent real world objects that the pipes must avoid and boundaries they must be contained within. The three primary obstacle shapes that PipeSynth can represent are half-space planes, parallelepipeds, and spheres. Each one can either block a network intersection with a wall or object, or can contain the network in a region like a duct or crawlspace. Obstacles are generally defined by an origin point, direction vector(s) (or radius), and whether they impose an interior or exterior boundary.

In order to evaluate if, and by how much, a pipe intersects an object, the obstacles are either externally or internally padded, depending on their constraining side. This inflated, or padded, distance is equal to the external radius of the pipe in question and some set obstacle clearance value. This technique allows for a much simpler intersection test of line segments and solids.

All half-space planes are defined by a point on their surface and their normal vector. With a normal vector defined to point outwards, there is an exterior penalty proportional to axial violation length of each pipe passing through it. During optimization, the optimization method will see this penalty and be forced to move each pipe back to the feasible side of this plane. These planes are useful for representing floors, ceilings, and other large areas that would constrict the design space. Planes are the simplest to test, as the point of intersection of a line

and a plane is a simple calculation as given by equation 29. If there is an intersection, the constraint returns a value proportional to length of the line segment that is on the violation side of the plane.

Parallelepipeds are located by a corner origin and scaled and oriented by three edge vectors that leave from this origin. These objects can either have an interior or exterior violation for each pipe in the network. They are useful for representing a variety of systems, ranging from machines, pumps, tanks, to areas of avoidance such as maintenance or electronic panels, to walls, doors, and pathways, to containment areas such as rooms or equipment zones.

Parallelepipeds are tested in a similar fashion because they are constructed with a set of flat quadrilaterals. The intersection with a quadrilateral uses the same equation as the line to plane intersection test, with one extra step. Once the intersection has been found to lie in the infinite plane that is coplanar with the quadrilateral in test, then the point must be tested to see if it lies inside the quadrilateral. This can be done by testing to see which sides of the infinite planes that are coplanar with the four adjacent quadrilaterals are. This is found by the sign of the dot product between the normal of the plane and the vector that joins the plain and the point in question. If the sign is positive, the point lies on the same side of the plane as the normal does. Other quick checks are done to compare the endpoints of the pipe in test and the minimum and maximum coordinates to quickly determine if a pipe is absolutely outside the parallelepiped.

Spheres are represented by an origin point and a radius. They can be used to keep networks a safe distance away from a particular object. Some examples of areas that might require clearance are access panels, very hot and very cold objects that could negatively impact the pipe network, and objects that require operating space like valves and switches.

To see if an obstacle intersects with a sphere, the constraint only needs to find the minimum distance between the sphere's origin point, and the line segment. This can be found by equation 28, and by verifying that the point on the infinite line which is closest to the sphere's origin also lies on the line segment. If not, the shortest distance between the sphere's origin and each of the line segment's endpoints will be used instead. If this distance is less than the radius of sphere, plus the external radius of the pipe and the set obstacle clearance value,

then the pipe is intersecting the sphere. The constraint's return value will again be proportional to the length of the line segment that is on the side of violation of the sphere.

6 OPTIMIZATION

This chapter presents the various methods used in the topological and parametric optimization stages of PipeSynth. This is followed by a discussion on the approaches used to produce multiple Pareto solutions from PipeSynth's two objective functions. Finally the integration of the objective and constraint functions into the optimization methods is reported.

The unique interaction between the two tiers of optimization in PipeSynth is presented in Figure 11.

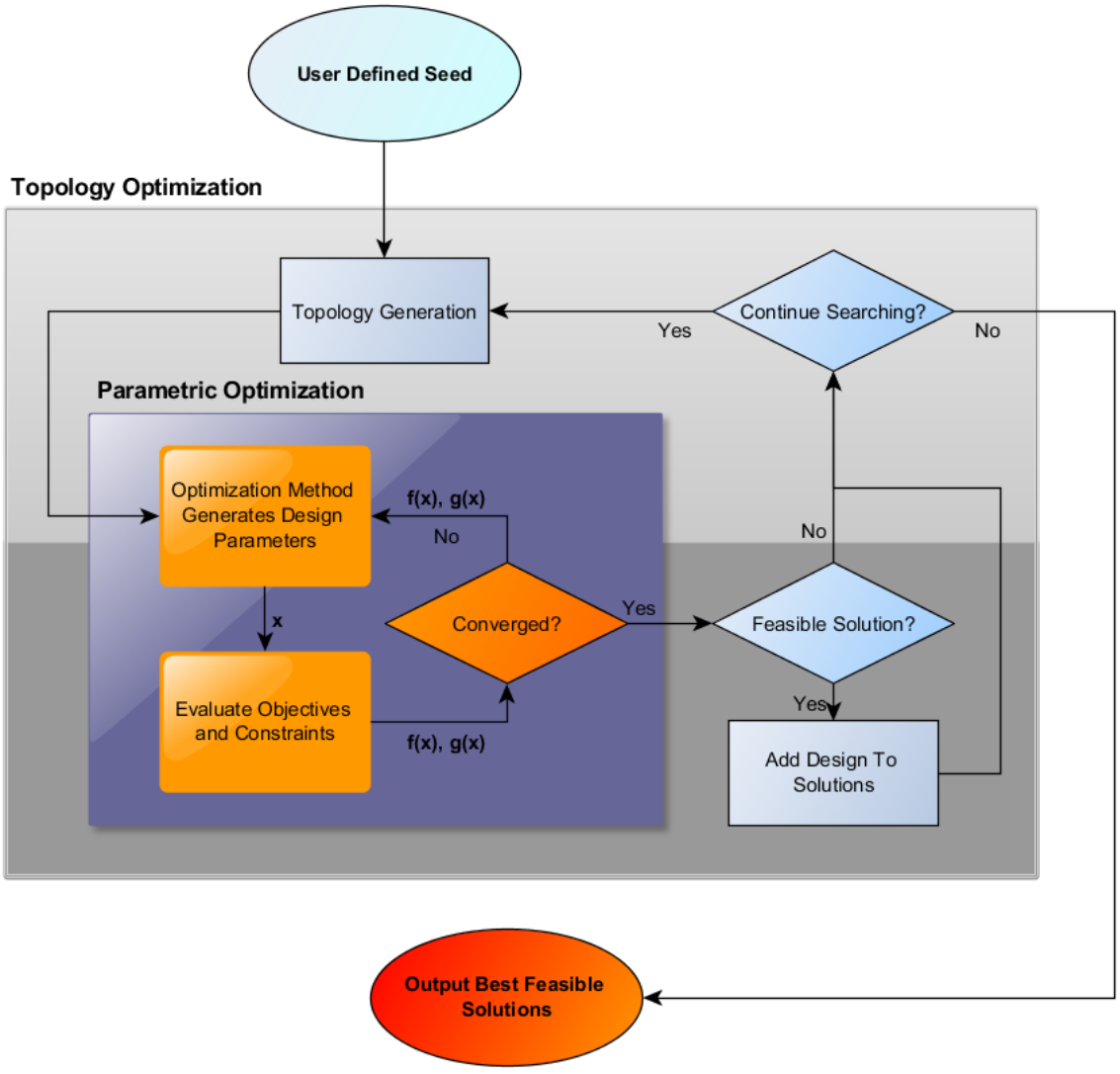

Figure 11. PipeSynth Optimization Procedures

Starting with a seed, a new topology is generated with the application of a grammar rule. This topology is then subjected to the parametric optimization subroutine. The parametric optimization method uses the evaluation techniques discussed in sections 4 and 5 to determine the fitness and feasibility of each solution candidate it generates. After the parametric optimization method converges, the design is tested for feasibility. If it passes, then it is compared to the list of best solutions found and added if it is a superior solution than those already found. Then if no stopping criteria are met for the topology optimization, a new topology is generated and the cycle repeats. The specific techniques and evaluating criteria of PipeSynth will now be discussed in the following subsections of this section.

6.1 Topology Optimization

PipeSynth primarily uses a uniform cost search for topology optimization. In the branched tree search space, every element in a level is searched, starting with the lowest transition cost and working towards the most expensive, before moving down to the next level of the tree. The process continues until no new solutions have been added to the Pareto front after a specified number of levels. If continually adding more pipes is not returning a better solution, then added even more pipes probably will only continue to add cost and head loss and never improve the network. The amount of levels it searches below the lowest Pareto solution should be scaled according to the number of channels (independent networks) because each channel needs one level each to add one additional pipe to its level.

With each grammar rule in PipeSynth representing a unique pipe type and size, there is one option corresponding to each grammar rule that has an inlet port size that matches the size of each open port. For example, a problem with four fittings available in one inch NPS and three fittings available in two inch NPS has a total of 7 grammar rules in the available ruleset(s). Then, if a candidate had one open one inch pipe and two open two inch pipes, then there would a total of 10 options (4+2*3).

If there were just four fittings, and thus four grammar rules, all for the same pipe size, then an example of the topology search on the seed in Figure 12 would follow the expansion of candidates seen in Figure 13.

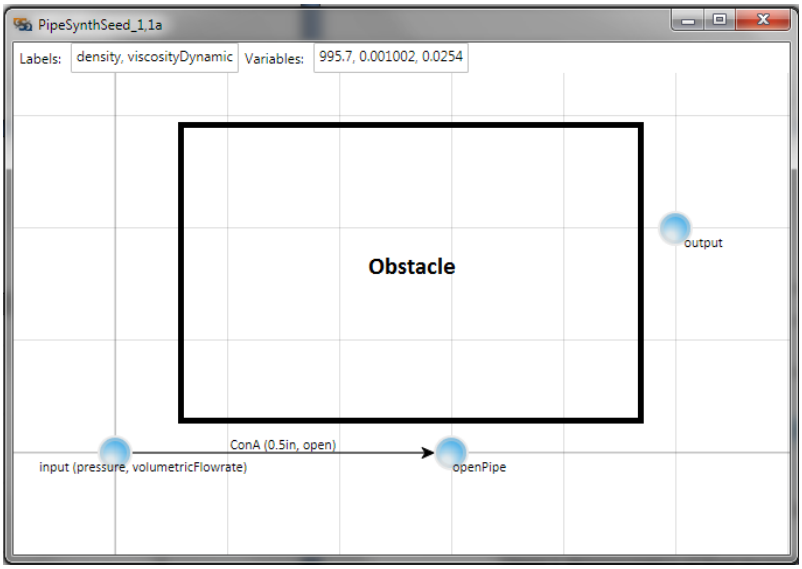

Figure 12. Simple example seed with an obstacle

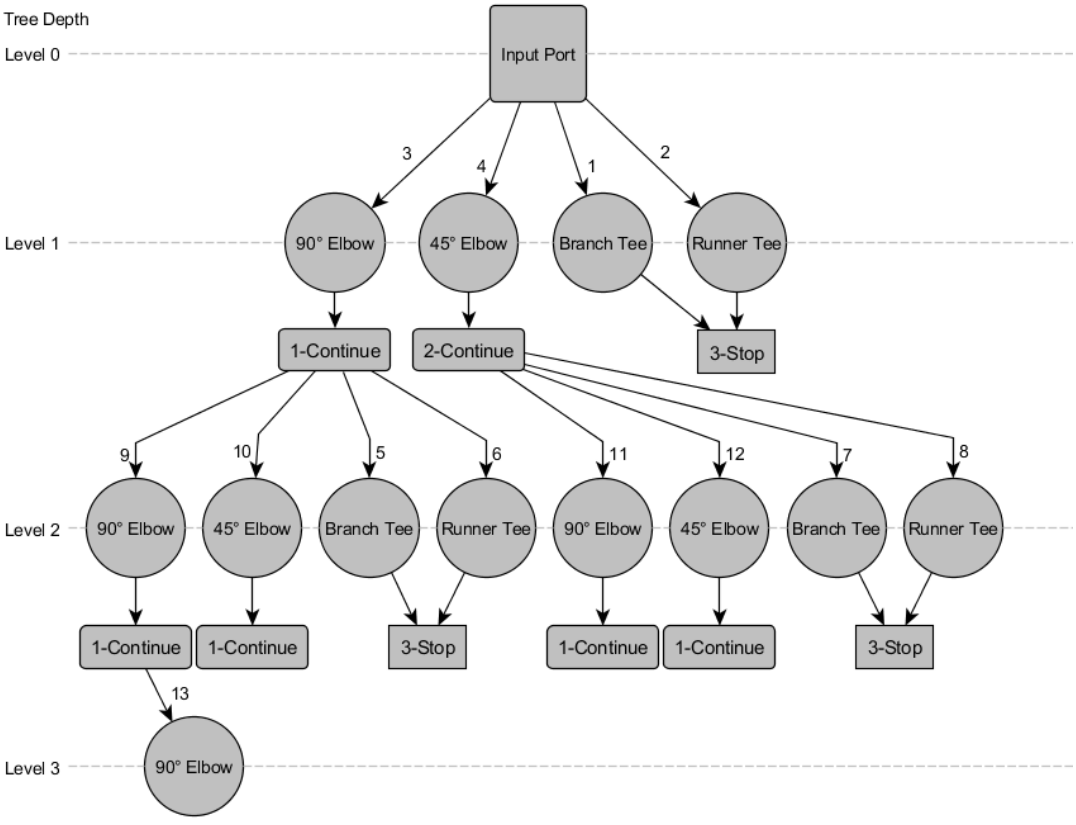

Figure 13. Example PipeSynth topology optimization of example seed of Figure 12

The numbered arrows in Figure 13 indicate the order in which candidates are searched. Uniform-cost-search selects the order based on the transition cost of each pipe at each tree level. This transition cost is based on the fitting type and size, with the smaller and more significant geometry changes having the lower costs. This problem contains just four fittings and prefers to use tees over elbows because they expand the number open pipes, and thus the possibility of connecting to multiple outlets with fewer fittings. After each candidate is parametrically optimized, the branch is determined to be at one of three conditions. Condition 1 represents that a feasible solution has been found, but the branch will continued to be searched for better solutions. Condition 2 is determined to be an infeasible solution, but it could lead to a feasible solution with more fittings. Therefore, this branch will also be kept and searched further. Condition 3 is termination to a branch and occurs when a stopping condition is met.

The seed in Figure 12 would become feasible with the addition of a 90° elbow, but not with a 45° because the 45° elbow fitting cannot create a network that will reach the outlet and avoid the obstacle. When a particular channel in a candidate has branched to contain more open pipes than outlets, the branch stops at Condition 3. This state also dictates that the current candidate is infeasible and that adding more pipes will never make it feasible again. Consequently, the branch is eliminated and does not expanded any further. Condition 3 is a direct result of PipeSynth being designed to create only branched type networks, and as a result, there is no way for additional fittings to merge open pipes back together to decrease the number of open pipes.

After parametric optimization of a candidate is complete, all the children are generated for that candidate and added to that sorted list, including an additional cost proportional to the depth of the tree that the child resides on. Thus, all the candidates/children are searched at one level before moving on to the next. The quality of each of these candidates as a feasible and better solution is determined during the parametric optimization stage.

6.2 Parametric Optimization

With a particular candidate selected with a fixed topology, the number of fittings and the order in which they are connected is known. The parametric optimization attempts to adjust them in a way to produce the best feasible solution possible. As described in representation section 4.1, each pipe length, α, and fitting attachment angle, Θ, make up the set of design variables, $\vec{x}$, that describe the final layout of the network. By optimizing these design variables, the best possible network(s) for a given topology can be found.

PipeSynth uses a combination of cyclic coordinate search, Powell's Method, and random hill climbing to find the global optimum. The unique combination provides the benefits of speed, accuracy, and robustness from each of the three methods. The implementation of these methods is shown in the flowchart of Figure 14.

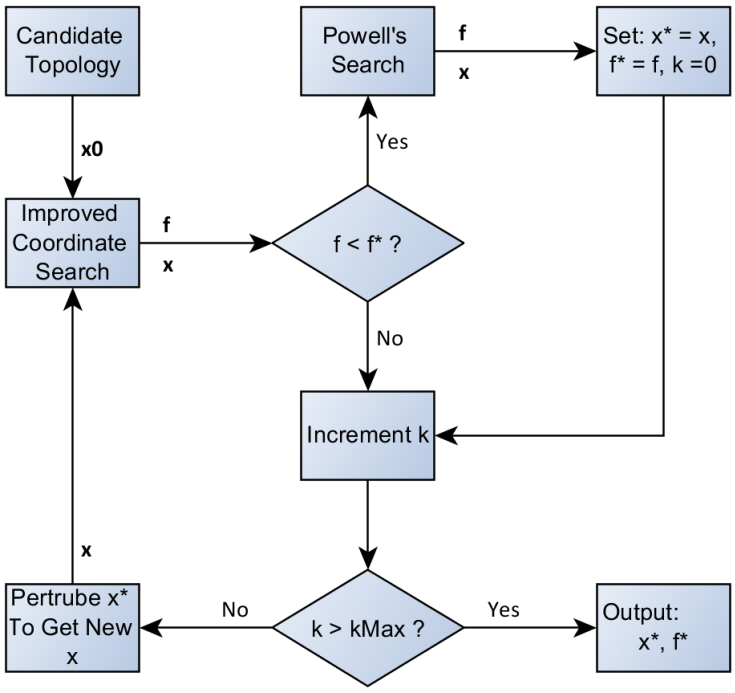

Figure 14. Parametric optimization flowchart

Cyclic coordinate search is used first because it is very quick at making a big improvement to $\vec{x}$, and it can be used quickly to test if a point is in a region with a better minimum than the best known minimum. Unfortunately, this method gets stuck easily at local minima and saddle points. Actually, it was found for some problems that in addition to

searching in the direction of every coordinate, searching along direction of every pair of coordinates had a significant improvement on avoiding local minima. This diagonal search is labeled improved coordinate search and not only searches in the positive direction of the unit vectors, but also every combination of positive and negative unit vectors for every pair. The performance improvement is a direct result of the way PipeSynth represents the networks from the parametric design variables. By increasing one variable and decreasing another at the same rate, it allows the length, α, (or angle of attachment, Θ) of one pipe to increase while the length (angle) of an adjacent pipe decreases. This allows fittings with a pair of collinear ports to "slide" or "spin" without affecting the design of the rest of the network during the parametric optimization stage.

Every time after a new best function evaluation has been found, Powell's method is used to get near the bottom of the current minima. Powell's method was found to fine tune $\vec{x}$ better than coordinate search, but at the cost of being anywhere from three to eight times computationally slower than cyclic coordinate search.

If a better point is not found with Coordinate Search, and if k is still less than kMax, after incrementing it as seen in Figure 14, then a random disturbance is added to the current best $\vec{x}^*$. This is done by selecting a random disturbance and attempting to get from the current best point over a hill and into a deeper valley. After this disturbance is added, the cyclic coordinate search method is used to quickly test $\vec{x}$ again. If the disturbance successfully found a significantly deeper valley, then the objective function's value will be the best found so far. This disturbance and improvement cycle continues until either a better point is found and Powell's method is employed to optimize it, or the limit of disturbance attempts, kMax, is reached. In order to adequately sample the space around x, the integer kMax is proportional to the number of design variables.

This unique multi-method approach proved to be more robust than any of three methods individually. Nelder-Mead's simplex method was also extensively tested and while it had a fast convergence rate, it failed to be any more efficient than cyclic coordinate search, and was especially unreliable at higher dimensions. The algorithm details of the parametric optimization come from the Object-Oriented-Optimization-Toolbox, OOOT, also created by Dr. M. I. Campbell and the members of the Automated Design Lab at the University of Texas at

Austin (24). Some modifications and new methods and features were added to this toolbox as a result of this research.

6.3 Pareto Optimality

A well-designed multi-objective optimization problem can be problematic because improving one objective often compromises another (21). Each problem in PipeSynth generates a two-dimensional Pareto curve that is comprised of the best trade-off solutions found for that application.

$$minimize(maximize)\ \vec{f}_{objective,total} = \{f_0(\vec{x}), f_1(\vec{x}), \dots, , f_m(\vec{x})\} \quad \textit{(33a)}$$

$$\textit{subject to } \vec{g}(\vec{x}) \leq 0,\ \vec{h}(\vec{x}) = 0 \quad \textit{(33b)}$$

A simple way of managing multi-objective problems is minimize or maximize the total sum the objectives—taking care to note the sign for each individual object which will determine if the individual objectives will be minimized or maximized. Each objective function would typically have an additional coefficient to balance inconsistent units between the different objects and to weight the importance of each objective. A linear sum multi-objective problem would take the form of equation 34.

$$minimize\ (maximize)\ \ f_{objective,total} = \sum_{i=1}^{n_{objectives}} \left(W_i * f_i(\vec{x})\right) \quad \textit{(34)}$$

Where $n_{objectives}$ is total number of objectives functions and W_i and $f_i(\vec{x})$ are the ith objective weight and objective function respectively. This may work for easier problems, especially with objectives that have consistent units, and where optimization can be executed several times to find the ideal $W_i's$. Using the linear sum method, while easy to implement and computationally efficient, often provides a poor distribution of solutions due to wildly varying ratios of objective functions for different problems and topologies. The way multiple objectives in PipeSynth were managed was by using a targeting method. This approach, equation 35, only minimizes one objective while the other is set to as a target constraint.

$$minimize\ (maximize)\ \ \vec{f}_0(\vec{x}) \quad \textit{(35a)}$$

$$subject\ to\ \{f_0(\vec{x}), f_1(\vec{x}), \dots, , f_m(\vec{x})\} \leq \vec{C}, \vec{g}(\vec{x}) \leq 0,\ \vec{h}(\vec{x}) = 0 \qquad (35b)$$

This is only applied during the parametric optimization, because the topology optimization is not controlled with continuous design variables. To apply the targeting method, PipeSynth first finds the extrema of the Pareto curve by minimizing each of the objectives independently, while leaving the alternative objective unconstrained. This finds the least expensive and least restrictive feasible solutions for each topology, as seen in Figure 15.

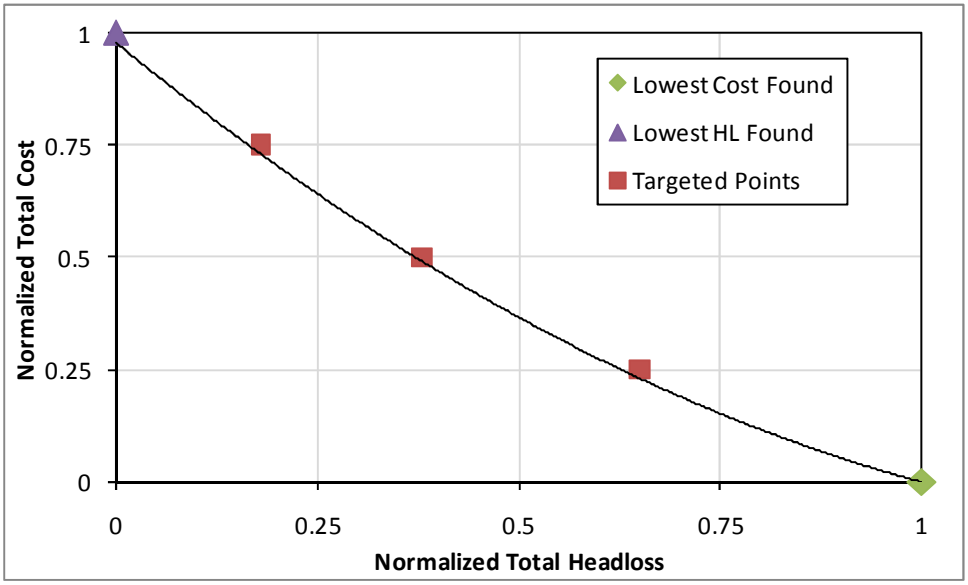

Figure 15. Targeting method of Pareto solutions during parametric optimization

The example in Figure 15 shows an even distribution of three cost targets to help approximate the Pareto curve by generating three additional points. To generate these new targeted solutions, each topology's head loss objective is parametrically optimized at each of those cost targets, as shown in equation (37).

$$min\,imize \quad f_{head\ loss}(\vec{x}) \qquad (36a)$$

$$subject\ to\ g_{cost} = f_{cost,total}(\vec{x}) - targetCost_i \leq 0 \qquad (36b)$$

The new inequality constraint, g_{cost} maintains the total cost below the targeted cost. Therefore $\vec{x}$ must minimize the head loss but without making the total network cost exceed the targeted

cost. By incrementing the target cost, several Pareto solutions can be generated for a specific topology. This method does not guarantee an even distribution of points along the Pareto curve but is very robust because the distribution will be generated regardless if the curve is concave or convex. It is also efficient because every parametric optimization returns a new useful point to aide in the approximation of the Pareto curve. Some methods have the ability to generate the Pareto curve in much finer detail and with a better distribution of points, but typically at the cost of requiring several more optimization executions. If a finer resolution of curve is desired, it easy to increase the number of cost targets per topology.

7 Results and Findings

Two different types of example networks are presented and discussed in this section. These are created to showcase the capabilities of PipeSynth for handling multiple channels, ports, and obstacles. Robustness and flexibility were placed above performance during the design and development of the algorithms used in PipeSynth. Therefore, each example problem has a specific scope to demonstrate the capabilities and allow thorough conclusions, while functioning under a reasonable amount of computational resources. It is possible to extend to more complex problems, where additional computation power can be harnessed with multiprocessor techniques, such as cluster computing and dedicated super computers.

The first type of example network shows multiple, simultaneous network synthesis for three single input and single output channels. This is representative of many chemical and industrial processing facilities where two or more devices may require multiple fluid supplies to operate. The second example network type is for a single channel with one input and four output locations. This example uses a food processing fire suppressant as the transported fluid, typically used to extinguish grease fires, and is representative of commercial kitchens and industrial food processing facilities.

7.1 Plant Example with Multiple Separate Channels

This first example displays PipeSynth's ability to handle multiple, separate channels that each contain a different type of fluid. Each channel path is constrained by the two processing objects, a central obstacle, and the floor, all while contained within a bounding box. The two dimensional GraphSynth seed is shown in Figure 16.

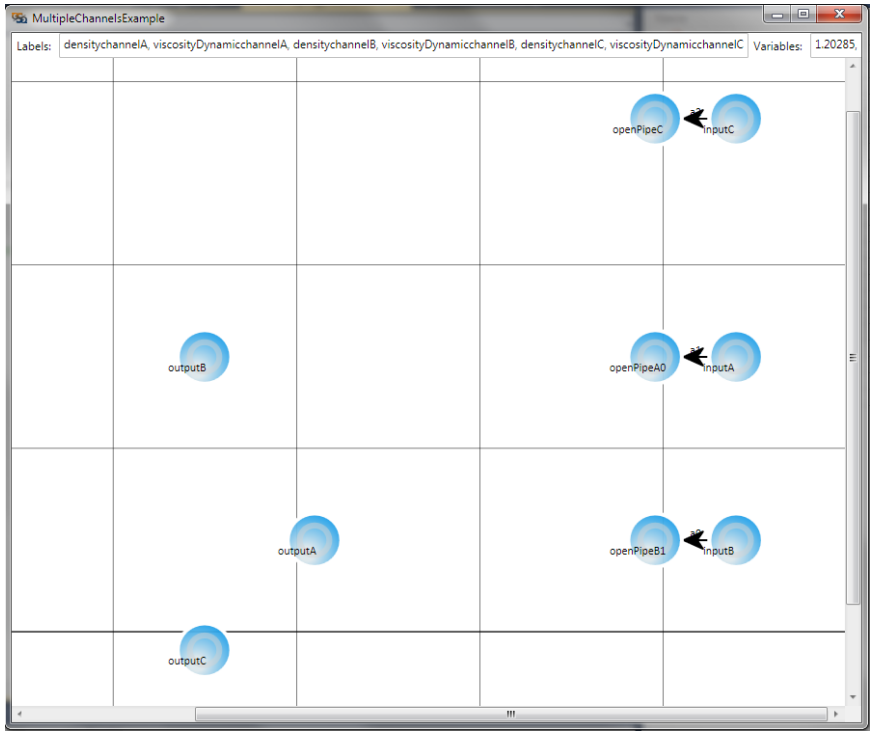

Figure 16. Plant example seed graph showing three independent channels

The varied input conditions, including the fluid properties, are shown below in Table 2.

Table 2. Fluid Properties and States for the three input channels of the plant example

		Density		Viscosity		Pressure		Flowrate		Pipe Dia
Channel	**Fluid**	**[kg/m^3]**	**[lbf/ft$^{3]}$**	**[Ns/m^2]**	**[cP]**	**[Pa]**	**[Psi]**	**[m3/s]**	**[GPM]**	**[NPS]**
A	Air	1.20285	0.075091	1.80E-05	0.018	200000	29	0.0003	4.755	3.00
B	Water	998	62.3031	0.001	1	400000	58	0.0001	1.585	0.50
C	SAE 10W oil	870	54.31233	0.104	104	200000	29	0.0003	4.755	1.25

The variety of fluids and flow conditions in the different channels display PipeSynth's flexibility at handling concurrent channel optimization for a wide variety of applications.

The test ran for approximately 39 hours and began finding solutions at level 7 of the tree. It proceeded through level 9 where it continued to find feasible solutions. The topology search terminated at the end of level 9 because no superior solutions had been found since level 7. The large tree was growing quickly and additional pipes would unlikely decrease both total cost and head loss below what was found at level 7. The resulting solutions are shown in Figure 17 below.

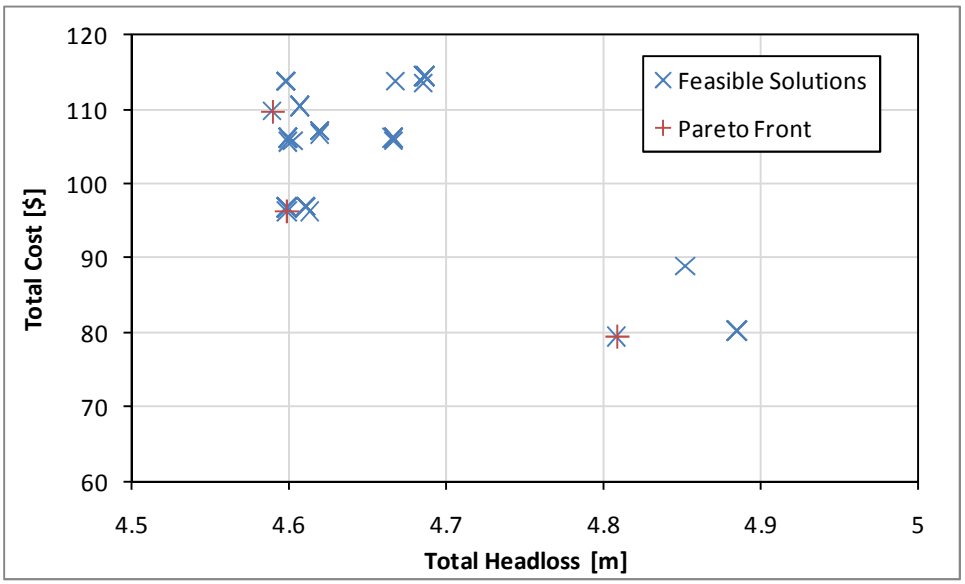

Figure 17. Pareto Plot of feasible solutions found for Plant Example

Three solutions from level 7 dominate the remaining 52 feasible solutions found. The objective values of these designs are shown in Table 4 of appendix A2. These solutions form the Pareto curve, but the scatter makes it difficult to predict the complete curve. The discontinuity in the Pareto curve is likely attributed to the discrete nature of the different topologies that produce a wide variety of unique solutions. Each parametric optimization only returned one unique solution for each feasible topology. This shows that only allowing one pipe size per channel caused tradeoff solutions to occur only at the topological level.

The output graphs that represent the solutions at the extrema of the Pareto curve are shown in Figure 18 and Figure 20, respectively.

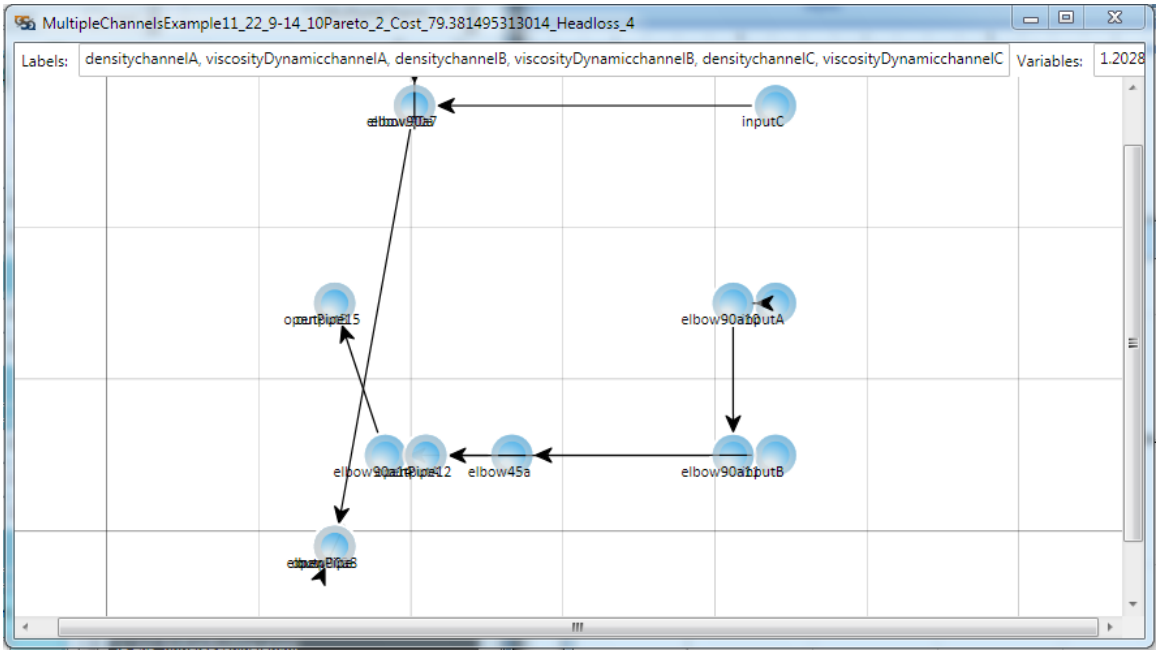

Figure 18. Plant example solution graph with lowest total cost

The three-dimensional models representing the networks and the objects were created with the assistance of the Helix 3D Toolkit, as seen in Figure 19 and Figure 21 (25).

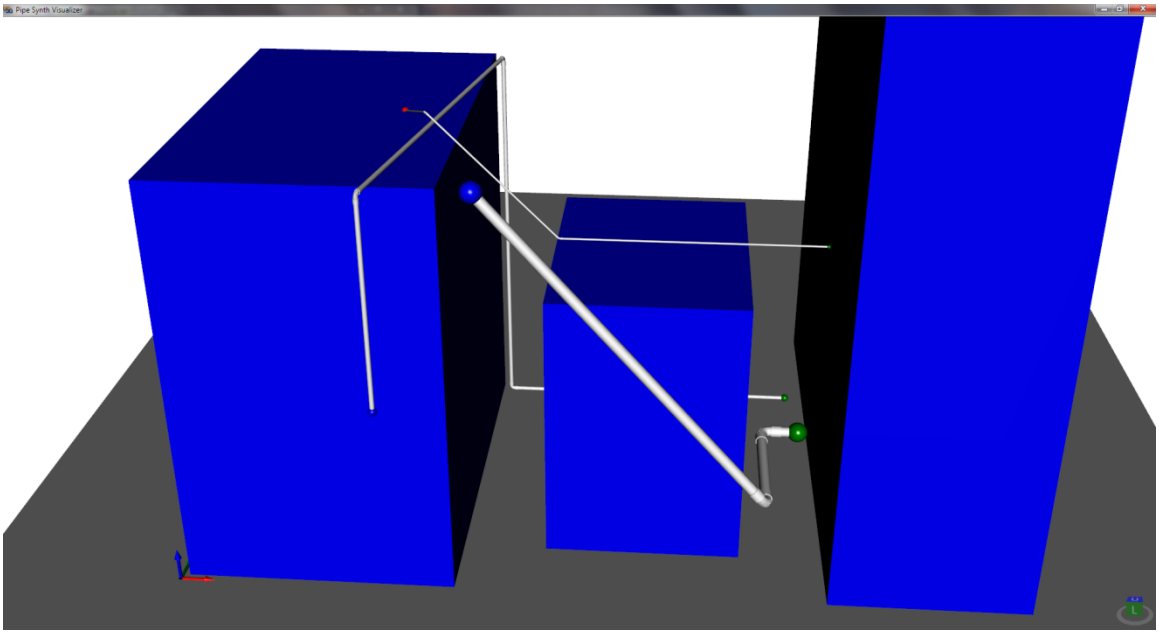

Figure 19. Lowest cost plant example modeled with Helix 3D Toolkit

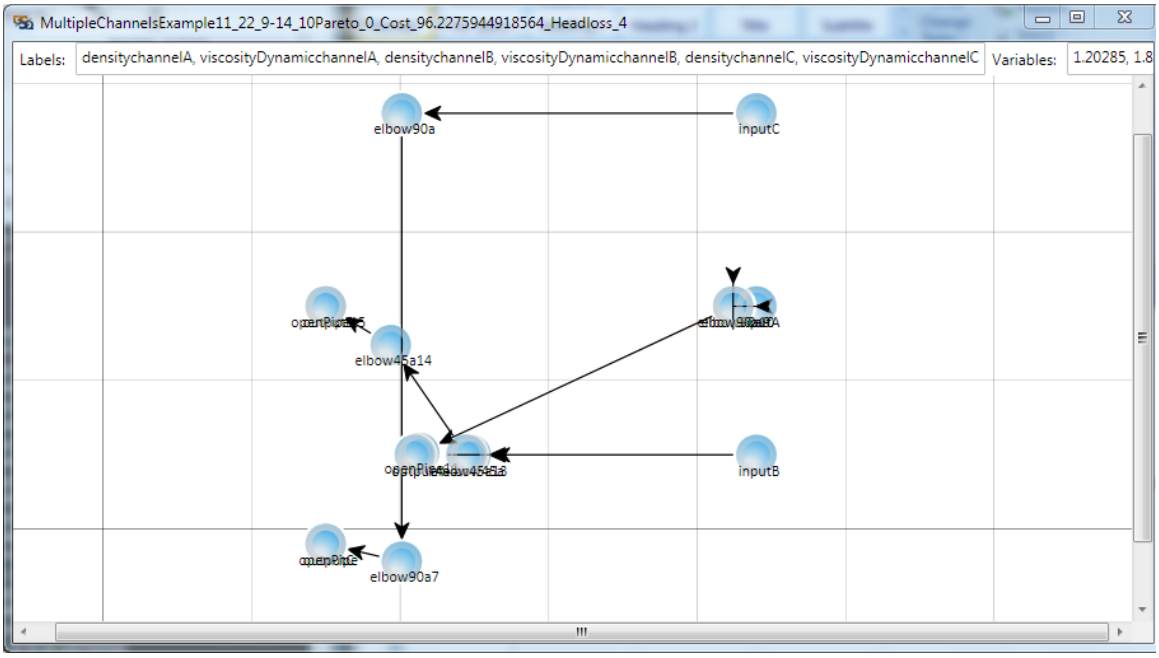

Figure 20. Plant example solution graph with lowest total head loss

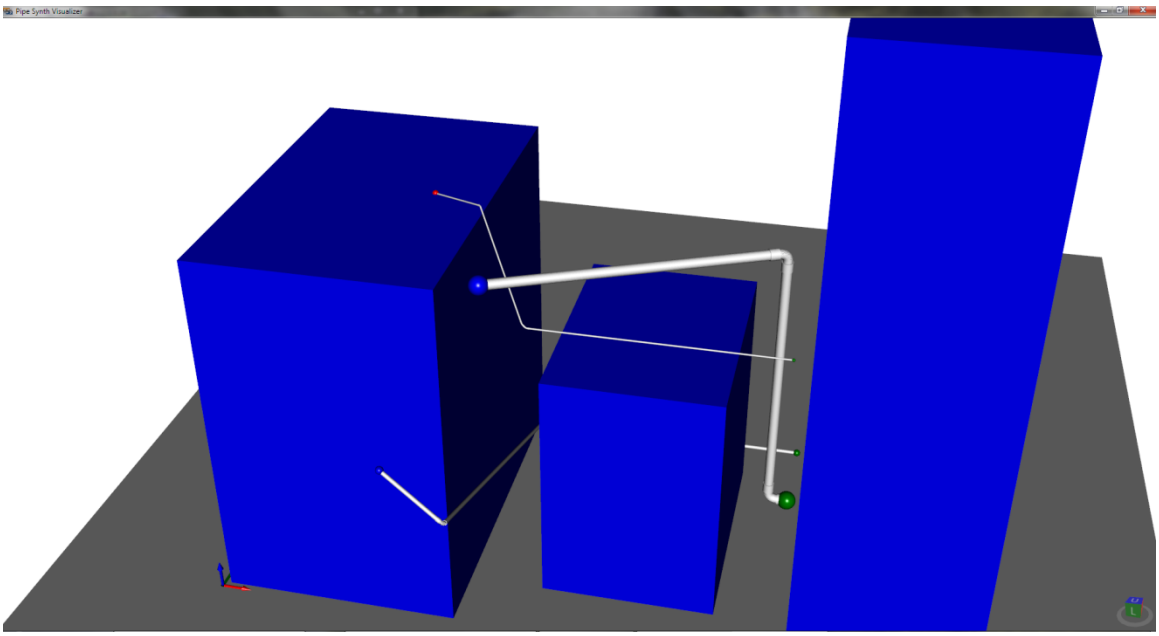

Figure 21. Lowest head loss plant example modeled with Helix 3D Toolkit

In each of these solutions, Figure 19 and Figure 21, the network designs correctly attach all the fittings and do not violate any constraints. While these solutions satisfy the design criteria, some of pipes appear to not be quite optimized to their full potential. For example, in Figure 19, one pipe travels up and over the left object, which seems intuitively unnecessary. However, the other two channels look very well optimized due to their direct routes. This shows that better designs may still be found with additional fine tuning to PipeSynth.

7.2 Food Processing Fire Suppression Example

The second demonstration is for the routing of a liquid fire suppressant to 4 workstations. The example liquid suppressant, PRX™, is produced by the ANSUL® company for grease and ventilation fires (26). It has fluid properties of the same order of water and is a suitable fluid for PipeSynth.

The example was setup with one input located near a wall and ceiling of a rectangular building with the need to provide suppressant to four separate workstations. The seed graph is shown in Figure 22.

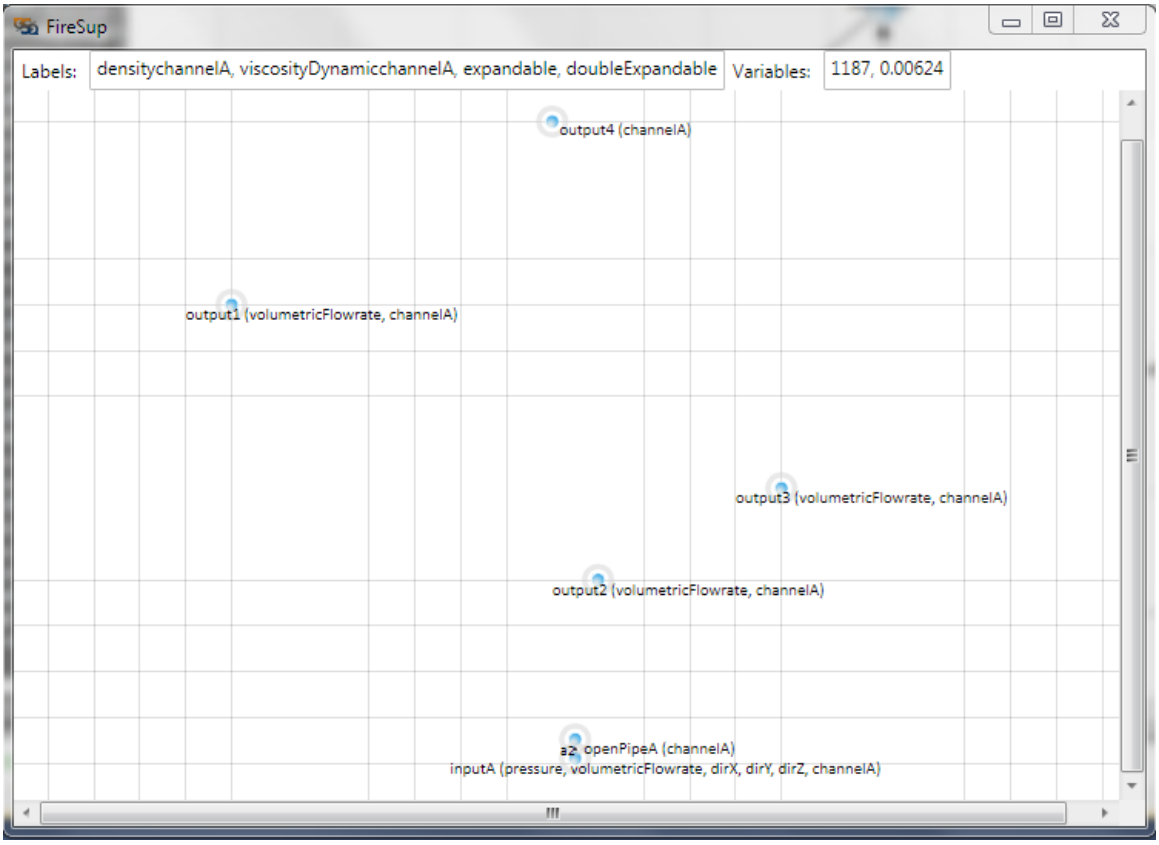

Figure 22. Fire Suppression seed graph with one input and four outputs

The input fluid properties and states were designed to match real world requirements and are listed in Table 3.

Table 3. Fluid Properties and States for Fire Suppression Example

		Density		Viscosity		Pressure		Flowrate		Pipe Dia
Port	**Fluid**	**[kg/m^3]**	**[lbf/ft^3]**	**[Ns/m^2]**	**[cP]**	**[Pa]**	**[Psi]**	**[m3/s]**	**[GPM]**	**[NPS]**
Supply	PRX™	1187	74.10199	6.25E-03	6.25	200000	29	0.0004	6.34	0.50
Outlet 1	PRX™	1187	74.10199	6.25E-03	6.25	Variable		0.0001	1.585	0.50
Outlet 2	PRX™	1187	74.10199	6.25E-03	6.25	Variable		0.0001	1.585	0.50
Outlet 3	PRX™	1187	74.10199	6.25E-03	6.25	Variable		0.0001	1.585	0.50
Outlet 4	PRX™	1187	74.10199	6.25E-03	6.25	Variable		0.0001	1.585	0.50

Four various sizes boxes were created to represent the different workstations requiring the suppressant and, while not displayed in the GraphSynth graphs, also act as obstacles to the network. The four outputs are placed on top of their respective boxes to allow connection to

each unit's independent fire suppression sprinkler system. To contain the network within the room, a bounding box is used. The results, Figure 23 and appendix A2, show how one topology found at level 4 of the tree dominated all the other designs found.

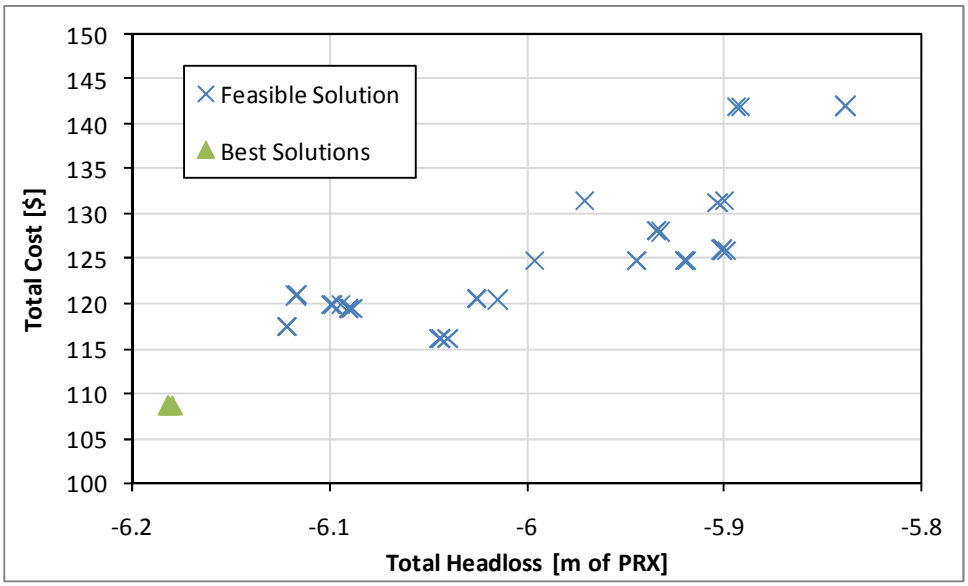

Figure 23. Pareto Plot of feasible solutions found for Plant Example

Again, the objectives were found to not be competing at the parametric level and always converged to one same point for each topology. The head loss is negative because the input is above all of the outputs and gravity reduces head loss to a negative level. The best topology is displayed in the output graph of Figure 24 and the three-dimensional model of this solution is shown in Figure 25.

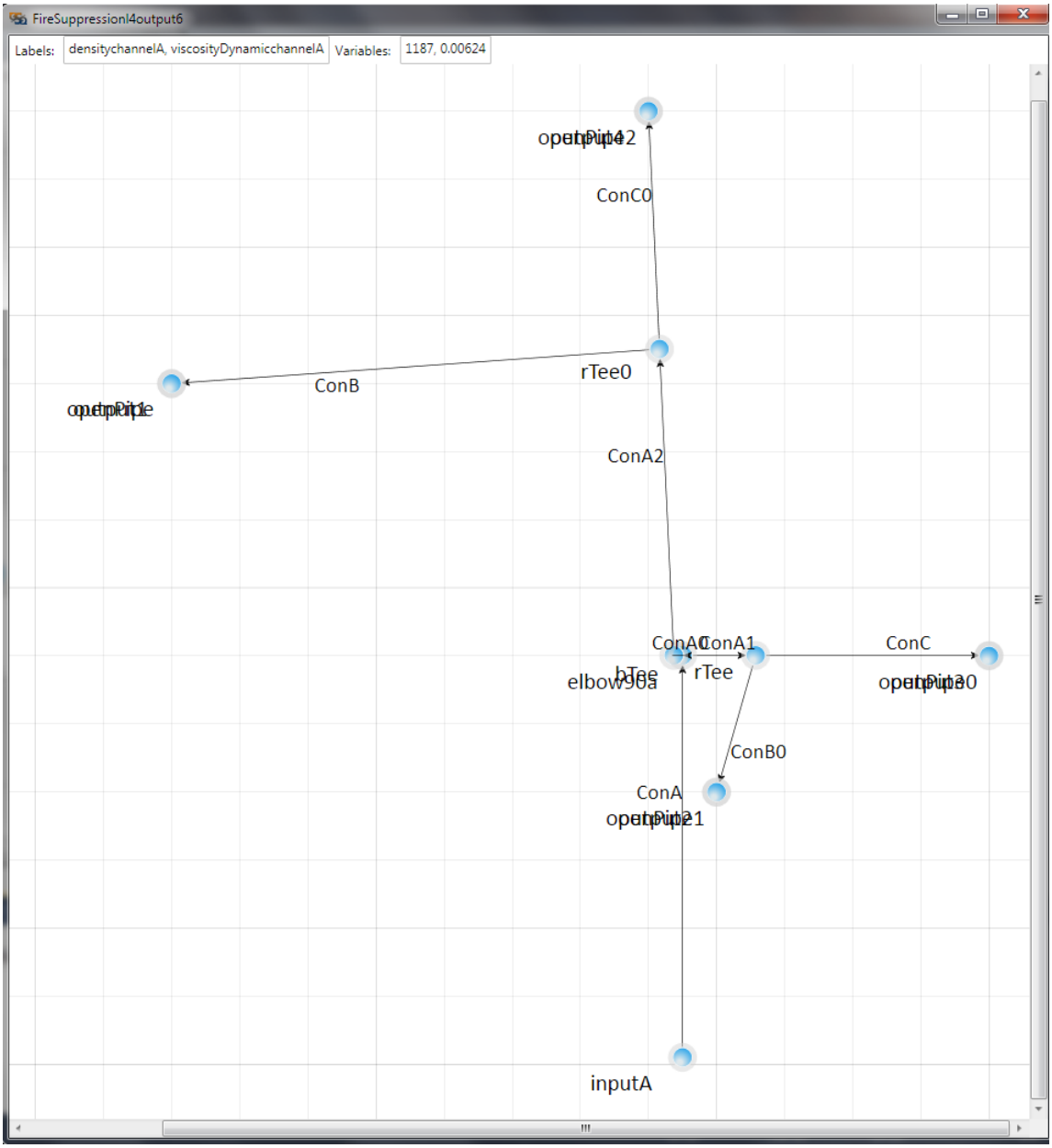

Figure 24. Best output solution graph for fire suppression example

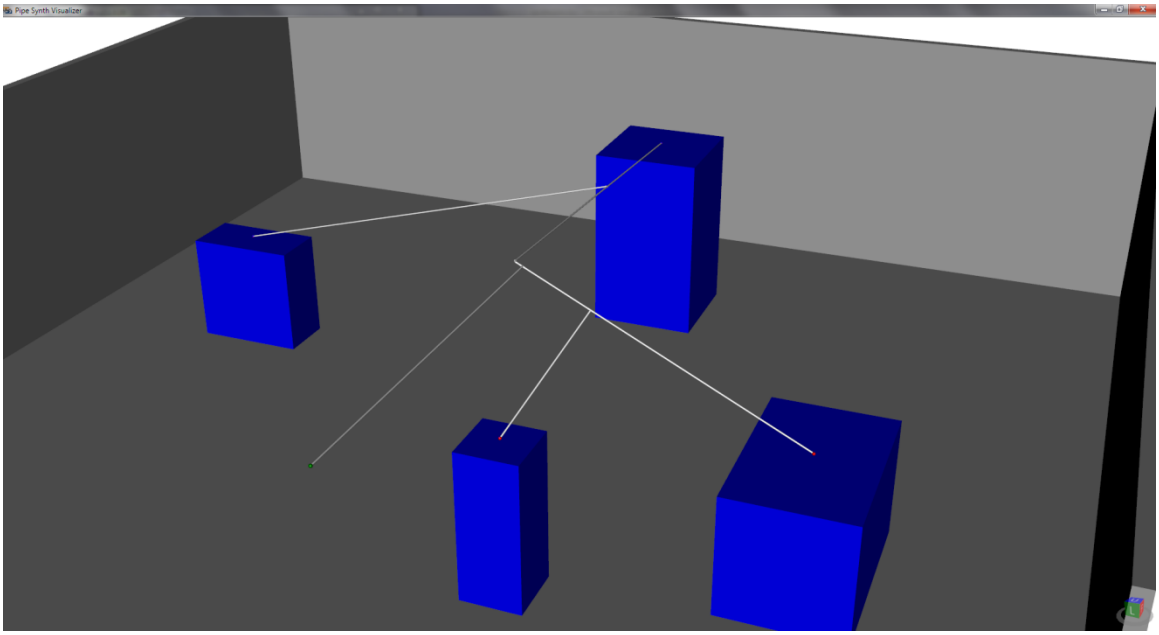

Figure 25. Best solution for fire suppression example modeled with Helix 3D Toolkit

The best solution, Figure 25, shows a branch tee (having its input port on the branched port side) and a 90° elbow as the first two fittings. The reason these fittings are used in conjunction instead of single tee is because the input pipe has its direction constrained to remain horizontal and therefore cannot reach the farthest output without a change of direction. This can be seen more clearly in Figure 26.

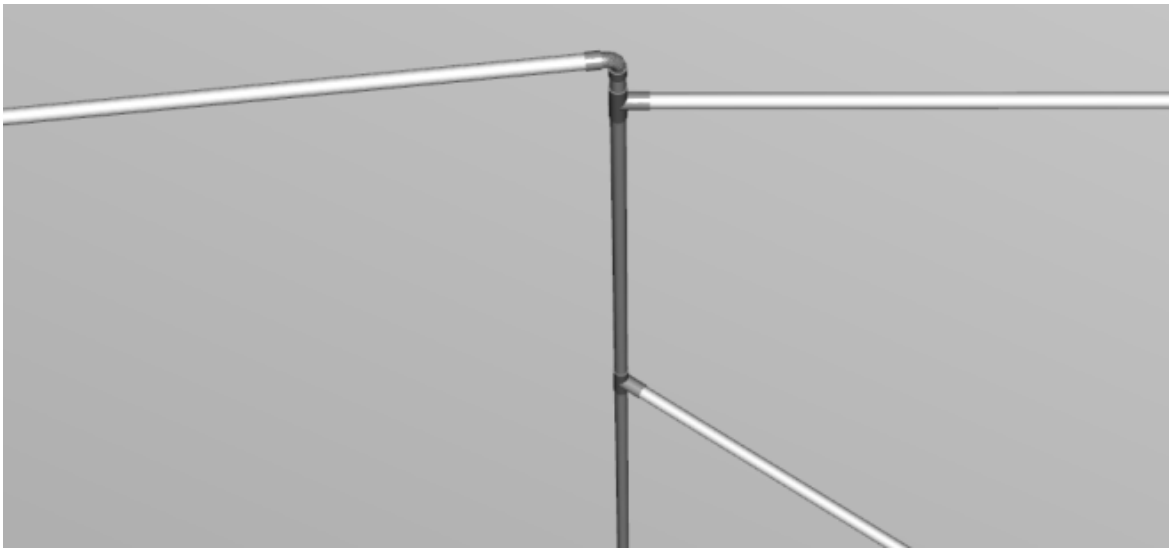

Figure 26. Tee and elbow fittings arrangement attached to input pipe in Figure 25

The horizontal pipe going exiting towards the right of Figure 26 attaches to the input port of the problem, while the angled pipe leaving towards the left points in the direction output port four. The pipes exiting the bottom attach to outputs two and three. In this solution, every pipe attaches properly to each fitting and each port, without violating any constraints.

The multiple outlets increased the number of branches significantly because with each multiple outlet fitting, a tee for example, arrives another location to apply the available grammar rules at. Therefore, the example was limited to only two elbows and two tees in order to limit the candidates in the tree. This allowed level 6 to be reached with 24 hours.

7.3 DISCUSSION

Several challenges and accomplishments were found by the study of this project. Increasing complexity became more and more resource intensive. Each additional level of the tree represents the addition of more pipe fitting. Each pipe fitting increases the number of design parameters by an amount equal to the number of ports the fitting contains. For example, an elbow adds one additional pipe length, α, and one additional angle of attachment, Θ, while a tee adds two additional pipe length design parameters along with its angle of

attachment. A plot of how much time was required for the average parametric optimization algorithm as a function of the amount of design variables is shown in Figure 27.

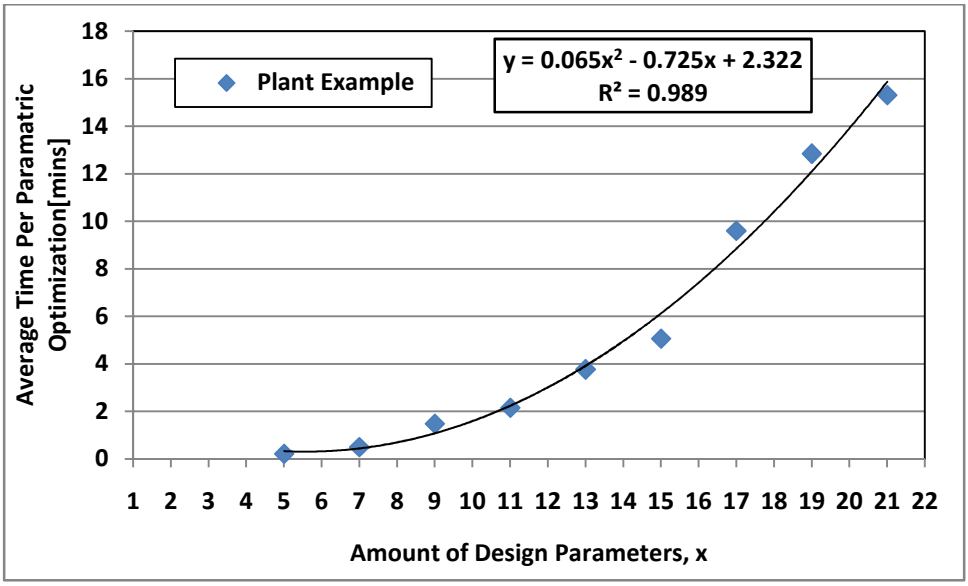

Figure 27. Average parametric optimization time as a function of design parameters

The plant example, section 6.1.1, has four obstacles and three separate channels. It was found that the amount of obstacles and channels added to the parametric optimization time considerably, and required much more fine tuning to produce good feasible solutions in a reasonable amount of time. Adding 7 fittings requires transversing to level 7 of the tree, which can be nearly impossible for large branching factors.

It was discovered that when a problem contains multiple channels, normal grammar rule application will produce confluent candidates. This can happen when, for example starting from the seed, rule 1 is applied to channel A, then rule 2 is applied to channel B. Then starting from the seed again, rule 2 could be applied to channel B, followed by rule 1 on channel A. To prevent time from being wasted searching these candidates with duplicate topologies, a data structure was created to keep track of every combination of rules that have been applied to generate candidates. However, it also takes into consideration which channels the rules were applied to, so that if a new candidate is created with a duplicate topology, it can be removed immediately.

It was also found that for some problems, the objectives may not be competing and therefore the Pareto curve approaches a single point. One reason this could be happening is because the parametric tuning only has control on the lengths of each pipe and angle of attachment of each fitting. Take a simple example of two straight points, joined by a coupling, joining an inlet and an outlet. If each pipe has the same diameter and linear cost per foot, then head loss and cost would no longer be competing. However, if we look at the alternative, and have two different diameter pipes, then the cost objective will try to maximize the ratio of the length of the smaller diameter, and therefore cheaper, to the larger diameter pipe. The head loss objective on the other hand, would then to minimize this ratio such that there is more of the less resistant larger diameter pipe.

In the attempt to find and produce the absolute best solutions for realistic pipe networks, the complexity of solvable problems was primarily limited by computational resources. Many previous optimization studies reduced the resource requirements by simplifying the problems with techniques such as discretizing the topology search space (13), or by limited the problems to 2 dimensions(13)(14)(19). PipeSynth allows pipes to freely move anywhere in three-dimensional space as their continuous variables are optimized. Mainstream computational performance improves by a factor of two about every 18 to 24 months.

One interesting modification to this problem would be to remove human decisions from this design process even further. Although very application specific, pressure drop could theoretically be used to directly calculate the pumping costs used to overcome them. Then this cost could be summed into the cost objective function, eliminating the head loss objective. The problem would then be a single objective problem with one true global optimum. PipeSynth would subsequently return the single least expensive feasible solution found, as opposed to a set of Pareto solutions. The benefits would be in the simplicity of having just one solution, and in the computational performance increase by eliminating the need to search for different trade-off solutions. The biggest drawback is the quality of the solution becomes even more sensitive to data provided by the user to define the problem (especially the new pumping cost function), introducing more uncertainty into the solutions found. When PipeSynth returns a variety of Pareto solutions, it gives the user a chance to throw away solutions that, while feasible, still may be undesirable. For example, a solution may have one stray pipe that travels

far away from any convenient structures for support. If PipeSynth has returned several solutions, rather than redefining the problem and searching for a new solution, the user can simply pick the next best solution that does not have a pipe in that undesirable location.

8 CONCLUSION

PipeSynth was quite successful in tackling many of the development goals and design considerations when designing pipe networks. It can handle a wide variety of fluids at a wide variety of flow conditions. It can build networks that conform to real world spatial constraints while retaining the correct fitting geometries between each and every pipe. With the computational resources available, Uniform Cost Search was an effective topological method when the optimal network resides within about the first 500 candidates. For more complex problems, stochastic tree search methods must be employed to transverse the tree adequately.

The use of penalty functions causes the n-dimension parametric search space to become very multi-modal. The most effective way of overcoming these local minima was the use of a random hill climbing algorithm. After each random climb, the improved cyclic coordinate search is used to see if the local minima has been escaped. After a better region is found, Powell's method is used to fine tune the design variables before returning the solution. This approach was found to be much more robust than Nelder-Mead, or any of three employed methods, used individually. The drawback to this approach was a much higher computational time during the optimization stage.

With PipeSynth's robust algorithms designed to handle increasingly more complex problems, and given adequate resources, PipeSynth should be able to solve much more complex, and much more significant, problems. This automated design and optimization study proves that complex, real-world, fluid networks can not only be solved with computational synthesis, but, given adequate resources, can also be optimized beyond what any human designer can create.

8.1 FUTURE WORK

Immediate future work is focused on improving the efficiency of the search process. By using more information about each type of problem more intelligently, the time wasted searching poor candidate and topologies for solutions can be significantly reduced. For example, more quick checks could be performed to see if the modification made to a previous candidate would actually allow a feasible solution to be produced. If a given topology could

never connect the ports regardless of any other constraints, then there is no point in trying to optimize it further.

There are several motivating features and capabilities that can still be added to PipeSynth. Additional cost considerations to more accurately model and assist in network design selection would be a great enhancement to any end user of this program. It is easier to construct and maintain fluid networks when the pipes are clustered and run parallel to adjacent pipes. Supporting pipes, especially larger ones, can be particularly important when heavy fluids are involved. When designed correctly, there will be an optimal tradeoff between minimizing material costs, head loss, and assembly (including bracketing) costs. Also, PipeSynth could assist in the design and optimization of the entire fluid network. Similar to the features available in commercial fluid network analysis tools, these could include the placement and sizing of pumps, tanks, heat exchangers, and valves. The fluid states at the subnodes located at the inputs and outputs of these devices would be related by some transfer function that relates the flowrate and pressure changes across each device.

There are several fluid flows that could be supported with additional representation and evaluation considerations. These include compressible fluids, flows through non-circular pipes/ducts, and fluids with significant temperature gradients. Compressible fluids no longer satisfy the conservation of volumetric flow used to solve flow states of the networks and would need significantly more equations and would likely need to be solved using numerical method techniques. Non-circular internal flows could be solved with additional steps in the calculation of the coefficient of skin friction. Many standard geometric duct shapes have analytical equations for finding the equivalent wall shear stress, and therefore skin friction. Shapes, without analytical stress equations, could be approximated using the concept of hydraulic diameter. This method equates a noncircular shape's wall shear stresses to that of an equivalent circular duct by calculating an equating diameter. However, the accuracy of this method is highly sensitive to the variation from a true circle (1). Finally, temperature gradients could be incorporated into PipeSynth, along with temperature dependent fluid properties, by having temperature as an additional state at each subnode. Then, the heat transfer could be calculated from these temperature states, the applicable heat transfer conduction and convection coefficients, and the ambient temperature outside the network.

APPENDIXES

A1. EXAMPLE GRAMMAR RULES

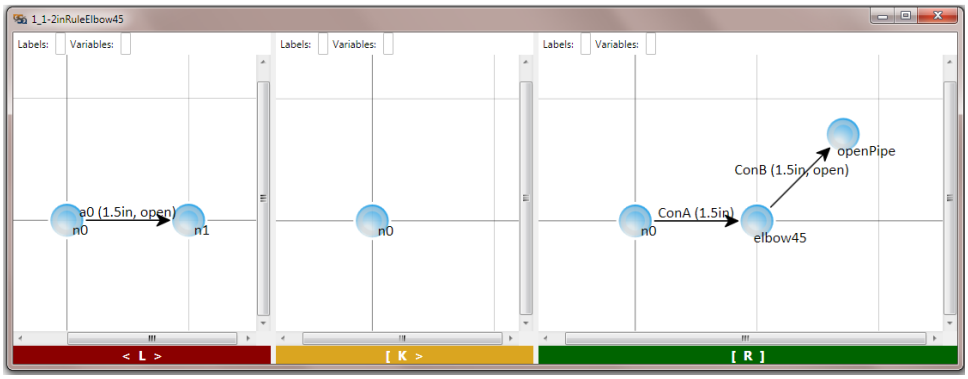

Figure 28. 1.5” NPS 45° elbow fitting

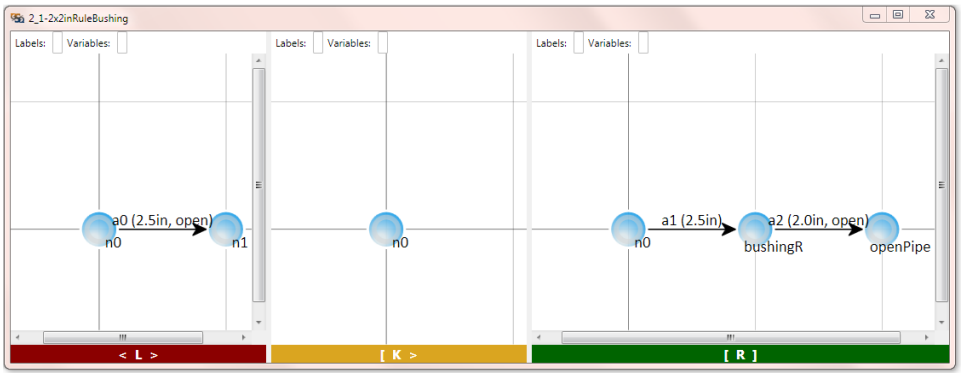

Figure 29. 2.5” to 2.0” NPS bushing fitting

Figure 30. 0.75” Tee with “runner” port input

A2. RESULTS DATA

Table 4. Plant Example Objective function values for feasible solutions found

All Feasible Solutions			
Headloss[m]	**Cost[$]**	**Headloss[m]**	**Cost[$]**
4.6112537	96.804877	4.6071397	110.36534
4.6112537	96.804877	4.6071396	110.36532
4.6112537	96.804877	4.5895755	109.74464
4.6112537	96.804877	4.6198829	107.11183
4.6112668	96.80524	4.6198829	107.11183
4.6138619	96.247445	4.6198829	107.11183
4.6864368	114.36628	4.6198911	107.11164
4.6864368	114.36628	4.6198735	107.11173
4.6864368	114.36628	4.6199212	106.63436
4.6864631	114.3659	4.5999905	106.05981
4.6864754	114.36577	4.5999905	106.05981
4.6681456	113.72303	4.5999905	106.05981
4.5990536	96.702846	4.5999905	106.05981
4.5990536	96.702846	4.5999649	106.05909
4.5990536	96.702846	4.5997963	105.53043
4.5990536	96.702846	4.8850022	80.176122
4.5990536	96.702937	4.8850022	80.176122
4.5990534	96.227594	4.8850022	80.176122
4.598344	113.67878	4.8084074	79.381495
4.598344	113.67878	4.6027446	105.65676
4.598344	113.67878	4.8520627	88.869404
4.598344	113.67878	4.66623	106.1291
4.598344	113.67878	4.66623	106.1291
4.6853549	113.45548	4.66623	106.1291
4.6071397	110.36528	4.66623	106.1291
4.6071397	110.36528	4.6659364	106.12587
4.6071397	110.36528	4.6667955	105.6423
Pareto Solutions			
Headloss[m]	**Cost[$]**		
4.5990534	96.227594		
4.5895755	109.74464		
4.8084074	79.381495		

Table 5. Fire suppression example objective function values for feasible solutions found

All Feasible Solutions			
Headloss[m]	**Cost[$]**	**Headloss[m]**	**Cost[$]**
-6.0253924	120.46103	-6.3863017	113.34909
-6.0253924	120.46103	-6.3868729	113.31196
-6.0253924	120.46103	-6.1212012	117.42789
-6.0144987	120.35043	-6.1212012	117.42789
-6.0899154	119.43857	-6.1211996	117.42789
-6.0899154	119.43857	-6.1212067	117.42796
-6.0899154	119.43857	-5.9009717	125.91517
-6.0885196	119.44896	-5.901031	125.91621
-6.0436735	116.03292	-5.9009639	125.91563
-6.0437498	116.0329	-5.8983482	125.88221
-6.0438035	116.023	-5.9443012	124.76095
-6.0397567	115.96552	-5.9443008	124.76094
-6.1163648	120.89092	-5.9443027	124.76096
-6.116365	120.89087	-5.9958031	124.71092
-6.116365	120.89082	-6.3539681	119.81803
-6.116885	120.84291	-6.3539681	119.81803
-5.9030794	131.20866	-6.3539669	119.81803
-5.9030794	131.20866	-6.3543112	119.76829
-5.9706184	131.38521	-5.9341652	127.98981
-5.8993366	131.36836	-5.9341454	127.98984
-5.9197471	124.77451	-5.9341208	127.98986
-5.9197471	124.77451	-5.9318469	127.9504
-5.9197471	124.77451	-5.8383929	141.96015
-5.9187872	124.73264	-5.8383929	141.96015
-6.0899154	119.43857	-5.8918026	141.8969
-6.0899154	119.43857	-5.8938555	141.81848
-6.0899154	119.43857	-6.0987681	119.80683
-6.0885196	119.44896	-6.0986593	119.80665
-6.3860282	113.3498	-6.0983695	119.80736
-6.3861615	113.34896	-6.0941788	119.77631
Pareto Solutions			
Headloss[m]	**Cost[$]**		
-6.18194249	108.7514898		

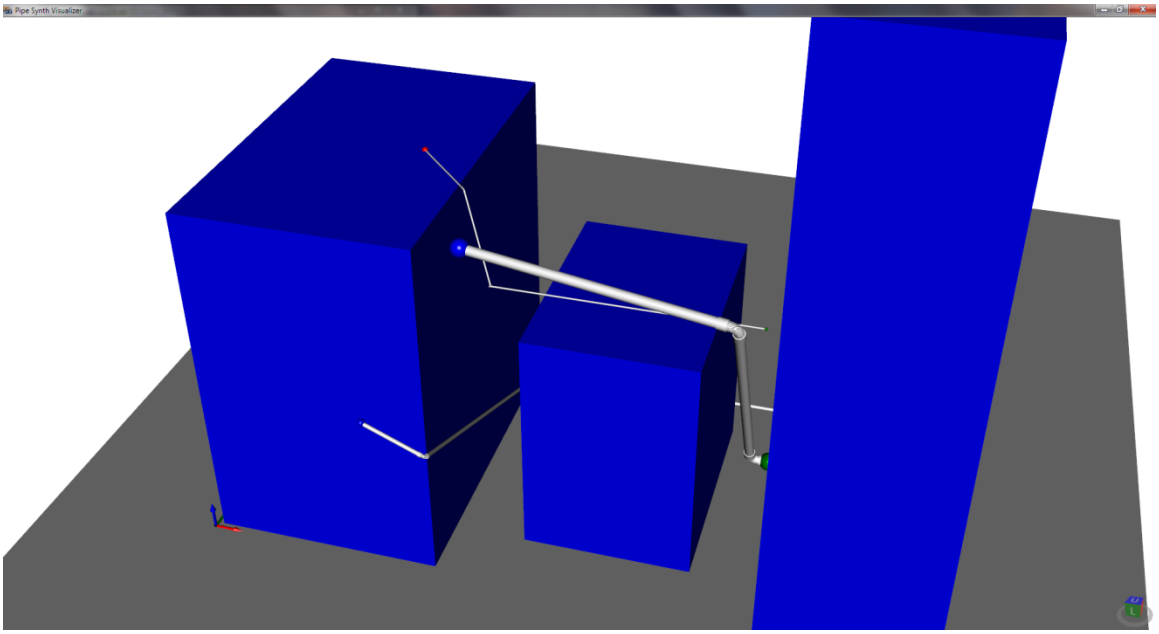

Figure 31. Second feasible solution for plant example found on the Pareto curve

REFERENCES

1. **White, Frank M.** *Viscous Fluid Flow.* Third Edition. NY : McGraw-Hill Education, 2006. pp. 89-111.

2. **GLS Software.** GLS Software. *GLS Software | Stellenbosh | South Africa | Products | Wadiso.* [Online] 2010. [Cited: 11 13, 2010.] http://www.gls.co.za/software/products/wadiso.html.

3. **Accutech.** *FLUIDFLOW3.* [Online] 2010. [Cited: 11 13, 2010.] http://www.accutech2000.com.au/Brochure.pdf.

4. **Bhave, Pramond R.** *Analysis of Flow in Water Distribution Networks.* Lancaster : Technomic Publishing Company, Inc., 1991. pp. 65-85.

5. —. *Optimal Design of Water Distribution Networks.* Pangbourn : Alpha Science International Ltd., 2003. pp. 16-33.

6. **Chanson, Hubert.** *Applied Hydrodynamics.* London : Taylor & Francis Group, 2009. pp. 29-30.

7. **Larock, E. Bruce, Jeppson, W. Roland and Watters, Z. Watters.** *Hydraulics of Pipeline Systems.* New York : CRC Press LLC, 2000. pp. 4-216.

8. **Nayyar, Mohinder L.** *Piping Handbook.* Seventh Edition. NY : McGraw-Hill, 2000.

9. **Stephenson, David.** *Pipeflow Analysis.* s.l. : Elsevier Science Publishers B.V., 1984. pp. 36-39.

10. **Vreugdenhil, Cornelis B.** *Computational Hydraulics.* Berlin : Springer-Verlag, 1989. pp. 1-7.

11. **Wood, W. L.** *Introduction to Numerical Methods for Water Resources.* New York : Oxford University Press Inc., 1993.

12. *Computing the Minimum Cost Pipe Network Interconnecting One Sink and Many Sources.* **Xue, Guoliang, Lillys, Theodore P. and Dougherty, David E.** [ed.] Siam J. Optim. 1, s.l. : Society for Industrial and Applied Mathematics, 1999, Vol. 10, pp. 22-42.

13. **Savic, Dragan A. and Walters, Godfrey A.** *Genetic Operators and Constraint Handling for Pipe Network Optimization.* School of Engineering, University of Exeter. Exeter, UK : s.n.

14. *Optimal Desgin of Offshore Natural-Gas Pipeline Systems.* **Rothfarb, B., et al.** s.l. : INFORMS: INstitute for OPerations Research, 1969.

15. *Optimization of a multiple reservoir system using a simulated annealing–A case study in the Mae Klong system, Thailand.* **Tospornsampan, Janejira, et al.** s.l. : Springer-Verlag, 2005, Paddy Water Environ.

16. *Topology optimization of flow networks.* **Klarbring, Anders, et al.** s.l. : Elsevier B.V., 2003.

17. **Pan, Tze-Chin and Kao, Jehng-Jung.** *GA-QP Model to Optimize Sewer System Design.* s.l. : JOURNAL OF ENVIRONMENTAL ENGINEERING © ASCE, 2009.

18. *Solving the Pipe Network Analysis Problem Using Optimization Techniques.* **Collins, M., et al.** 7, 1978 : Managment Science, Vol. 24.

19. *A parameter-free self-adapting boundary genetic search for pipe network optimization.* **Afshar, M. H. and Marino, M. A.** s.l. : Springer Science+Business Media, LLC, 2007.

20. **Campbell, Matthew I.** GraphSynth2. *GraphSynth.* [Online] 07 14, 2010. [Cited: 11 19, 2010.] http://www.graphsynth.com/.

21. **Belegundu, Ashok D.** *Optimization Concepts and Applications.* Upper Saddle River : Prentice-Hall, Inc., 1999.

22. **McMaster-Carr®.** McMaster-Carr® - Pipe Fittings. *McMaster-Carr®.* [Online] 11 24, 10. [Cited: 8 25, 10.] http://www.mcmaster.com/#pipe-fittings/=9v7qhe.

23. **Eberly, David.** *Distance Between Two Line Segments in 3D.* s.l. : Geometric Tools, LLC, 1999.

24. **Campbell, Matthew I.** Object-Oriented Optimization Toolbox (OOOT). *Codeplex.* [Online] Automated Design Lab at the University of Texas at Austin, 10 24, 10. [Cited: 11 24, 30.] http://ooot.codeplex.com/.

25. Helix 3D Toolkit. *CodePlex.* [Online] 11 21, 2010. [Cited: 11 24, 2010.] http://helixtoolkit.codeplex.com/.

26. **Ansul.** PRX Liquid Fire Supperssant. *Ansul Fire Protection.* [Online] Ansul, 2005. [Cited: 11 23, 2010.] www.ansul.com/AnsulGetDoc.asp?FileID=8348.

27. **Nayyar, Mohinder L.** *Piping Databook.* New York : McGraw-Hill, 2002. pp. 439-484,543-556.

28. **Bates, Paul D, Lane, Stuart N. and Ferguson, Robert I.** *Computational Fluid Dynamics: Applications in Enviromental Hydaulics.* Chichester : John Wiley & Sons Ltd, 2005.

29. **Fox, Robert W., McDonald, Alan T. and Pritchard, Philip J.** *Introduction to Fluid Mechanics.* Hoboken : John Wiley & Sons, Inc., 2006.

30. *Optimal dimensioning model of water distribution systems.* **Gomes, Heber Pimentel, et al.** 4, Paraiba, Brazil : Water SA, July 2009, Vol. 35.

31. *Reliability Based Design of Water Distribution Networks Using Multi-Objective Genetic Algorithms.* **Prasad, T. Devi, Hong, Sung-Hoon and Park, Namsik.** 3, s.l. : KSCE Journal of Civil Engineering, 2003, Vol. 7, p. pp. 351~361.

32. *Optimal Dimensioning of Pipe Networks with Application to Gas Transmission Networks.* **Wolf, Danil de and Smeers, Yves.** 4, s.l. : INFORMS, July 1996, Vol. 44, pp. 596-608.

33. **Murray, Glenn.** Rotation About an Arbitrary Axis in 3 Dimensions. *mines.edu.* [Online] Colorado School of Mines , 09 07, 2005. [Cited: 11 10, 2010.] http://inside.mines.edu/~gmurray/ArbitraryAxisRotation/ArbitraryAxisRotation.html.

34. *Reliability Based Design of Water Distribution Networks.* **T. Devi Prasad, Sung-Hoon Hong, and Namsik Park.** 3, s.l. : KSCE Journal of Civil Engineering, May 2003, Vol. 7, p. 351~361.